Sheet 7 (Sachin Gupta, Bhavesh Rajpoot and Simran Joharle)

In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
from IPython.display import Image

plt.rc('font', family='monospace', size=14, serif='courier')
plt.rc('mathtext', fontset='stix')
```

```
/opt/anaconda3/envs/mlph/lib/python3.10/site-packages/tqdm/auto.py:22: TqdmWar
ning: IProgress not found. Please update jupyter and ipywidgets. See https://i
pywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [ ]:
```
Image(filename='1.jpg')
```

Out[ ]:

1. (a) Binary logistic sigmoid function

$$\phi(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d\phi}{dx} = \frac{-1}{(1+e^{-x})^2}(-1) = \frac{1}{(1+e^{-x})^2}$$

$$\boxed{\frac{d\phi}{dx} = \phi^2}$$

1. (b)

$$\phi(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Hyperbolic tangent function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$= \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

$$= \frac{1 - e^{-2x} - 1 + 1}{1 + e^{-2x}}$$

$$= \frac{2}{1+e^{-2x}} - \frac{(1+e^{-2x})}{(1+e^{-2x})}$$

$$\boxed{\tanh(x) = 2\phi(2x) - 1}$$ ← Shifted value

↑ Scaled Binary Sigmoid

2 Log-sum-exp and soft(arg)max

(b)

In [ ]:
```
Image("2.1.jpg")
```

Out[ ]:

(2). LSE and Soft(arg) max.

$$\text{lse}(\vec{\sigma}, \nu) = \frac{1}{\nu} \log\left(\sum_{j=0}^{K} \exp(\nu \sigma_j) \right)$$

$$\text{Soft(arg) max}(\vec{\sigma}; \nu)_k = \frac{\exp(\nu \sigma_k)}{\sum_{j=0}^{K} \exp(\nu \sigma_j)}$$

$\nu \in \mathbb{R}^+$, $k = 1, \dots K$.

(a). for $\sigma_1 = (1,2,3)^T$ we have

$$\boxed{SM(\vec{\sigma}_1, \nu)_k = \frac{e^{\nu \sigma_k}}{e^{\nu} + e^{2\nu} + e^{3\nu}}}$$

(where $\sigma_k \in \{1,2,3\}$) — (2.1)

for $\sigma_2 = (11, 12, 13)^T$

$$SM(\vec{\sigma}_2, \nu)_k = \frac{e^{\nu \sigma_{2k}}}{e^{11\nu} + e^{12\nu} + e^{13\nu}} =$$

Clearly $\sigma_{2k} = 10 + \sigma_{1k}$

Hence, $$SM(\vec{\sigma}_2, \nu)_k = \frac{e^{10\nu} \cdot e^{\sigma_{1k}}}{e^{10\nu}\left(e^{\nu} + e^{2\nu} + e^{3\nu}\right)}$$

$$\boxed{SM(\vec{\sigma}_2, \nu)_k = \frac{e^{\sigma_{1k}}}{e^{\nu} + e^{2\nu} + e^{3\nu}}}$$ ——→ (2.2)

similarly for $\sigma_3 = (10, 20, 30)^T$

In [ ]:
```
Image(filename='2.2.jpg')
```

Out[ ]:

$$SM(\sigma_3, w)_k = \frac{e^{\sigma_{3k} w}}{e^{10w} + e^{20w} + e^{30w}}$$

$$\sigma_{3k} = 10\,\sigma_{1k}$$

thus $$\boxed{SM(\sigma_3, w)_k = \frac{e^{10\,\sigma_{1k} w}}{e^{10w} + e^{20w} + e^{30w}}} \quad (2.3)$$

from eq (2.1) (2.2) and (2.3) we conclude that.

$\underline{\underline{\sigma_1}}$ and $\underline{\underline{\sigma_2}}$ yield identical result.

For General Case

(1) Constant offset i.e $\sigma_k' = \sigma_k + \alpha$

where α = Constant offset

for $\vec{\sigma}_k$, $$SM(\sigma_k, w)_k = \frac{e^{\sigma_k w}}{\sum_{j=0}^{k} \exp(w\sigma_j^\circ)} \quad (2.4)$$

for $\vec{\sigma}_k'$, $$SM(\sigma_k', w)_k = \frac{e^{\sigma_k w} \times e^{\alpha w}}{\sum_{j=0}^{k} \exp(w\sigma_j^\circ + w\alpha)}$$

$$SM(\sigma_k', w)_k = \frac{e^{\sigma_k w}}{\sum_{j=0}^{k} \exp(w\sigma_j^\circ)} = SM(\sigma_k, w)$$

Hence Soft max is invariant under Constant offset.

In [ ]:
```
Image(filename='2.3.jpg')
```

Out[ ]: (ii) rescaling of it's input i.e. $\sigma_k' = \beta\sigma_k$

$$SM(\sigma_k', \nu)_k = \frac{\exp(\beta\sigma_k\nu)}{\sum_{j=0}^{K}(\beta\sigma_j\nu)}$$

which does not give us back equation (2.4)
thus under rescaling of input softmax is not invariant.

7.d

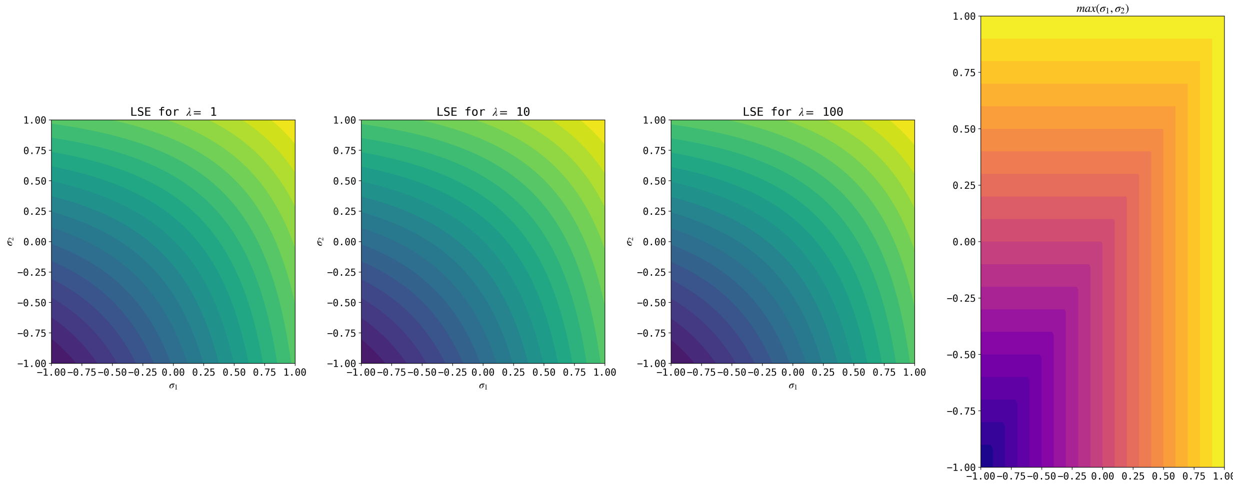
$$lse(\vec{\sigma}, \nu) = \frac{1}{\nu}\log\left(\sum_{j=0}^{K}\exp(\nu\sigma_j)\right)$$

diff w.r.t. 'σ_k'

$$\frac{d\,lse}{d\vec{\sigma}_k} = \frac{1}{\nu}\frac{\nu\exp(\nu\sigma_k)}{\sum_{j=0}^{K}\exp(\nu\sigma_j)}$$

$$\boxed{\frac{d\,lse}{d\sigma_k} = \frac{\exp(\nu\sigma_k)}{\sum_{j=0}^{K}\exp(\nu\sigma_j)} = \text{Soft (arg) max}_k.}$$

In [ ]:
```
def logsumexp(x, lamb):
    """ input
    x- is a meshgrid vector obtained by the command np.meshgrid(x,y)
    output - log sum exp

    """

    # TODO: implement the logsumexp

    lse =  np.log(np.sum(np.exp(lamb*x),axis = 0))/lamb  ## since x is a 3d vec
    return lse


# TODO: set up a grid of points in [-1, 1] x [-1, 1]
xlist = np.linspace(-1, 1.0, 100)  ##creating xlists and ylists from -1 to 3 ar
ylist = np.linspace(-1, 1.0, 100)   ##creating ylists as above
sigmas = np.meshgrid(xlist, ylist)  ##creating meshgrid


# TODO: I recommend you set up a function to set up an Axes object with the cor
#       equal aspect and maybe x and y ticks.
def set_up_axes(ax):

    ax.set_xlabel(r'$\sigma_1$',size = 15)
    ax.set_ylabel(r'$\sigma_2$',size = 15)
    ax.set_aspect('equal')


# TODO: calculate and plot the functions as specified in the task
sig1 = sigmas[0]
sig2 = sigmas[1]

fig,axs = plt.subplots(ncols = 4,figsize = (25,10),dpi=300)

lamb_array = [1,10,100]

for i in range (len(lamb_array)):
    lamb = lamb_array[i]

    lse = logsumexp(sigmas,lamb = lamb)

    axs[i].contourf(sigmas[0],sigmas[1],lse.reshape(sigmas[0].shape),20)
    axs[i].set_title(r'LSE for $\lambda = $ ' + str(lamb))
    set_up_axes(axs[i])

##function for max(sig1,sig2)
max_s1s2 = np.max(sigmas,axis = 0).reshape(sigmas[0].shape)
axs[3].contourf(sig1,sig2,max_s1s2.reshape(sigmas[0].shape),20, cmap = 'plasma'
axs[3].set_title(r'$max(\sigma_1,\sigma_2)$')

fig.tight_layout()
```

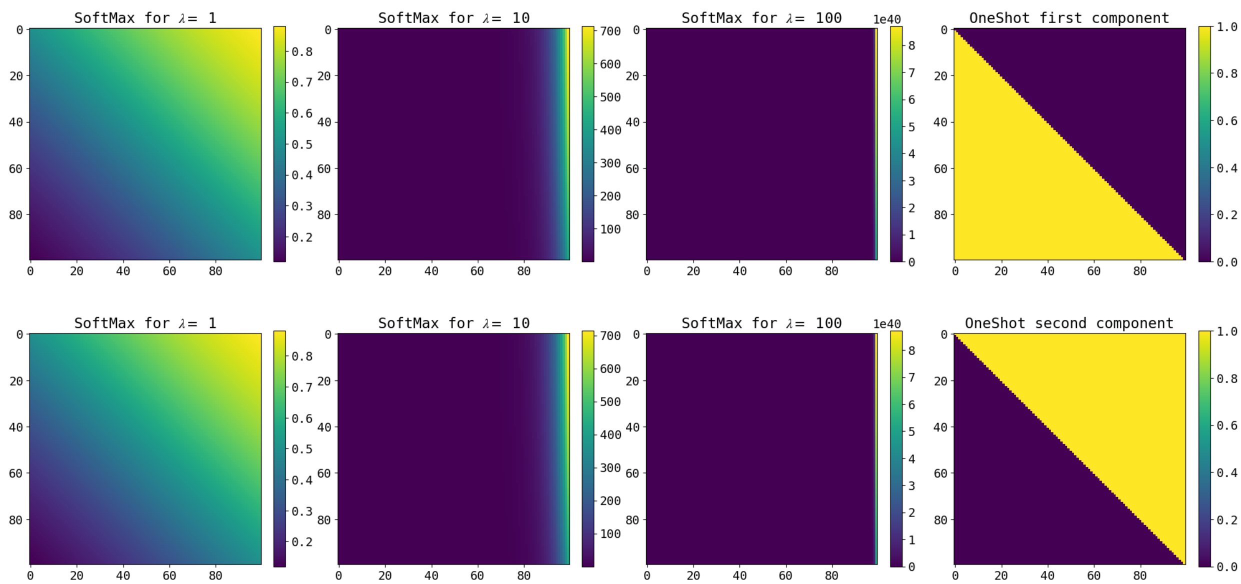


In [ ]:

```
def softmax(x, axis, lamb=1):
    """
    input :
    x is a meshgrid vector obtained by the command np.meshgrid(x,y)
    axis - represents the component of the soft max ie either sigma 1 or sigma2
    output :
    components of a softmax for all the grid points for a particular axis
    """

    # TODO: implement the softmax function. Axis should specify along which axi
    denominator_sum = np.sum(np.exp(lamb*x),axis = 0)  ##finding the sum of sof
    numerator =  np.exp(lamb*x[axis])  ##finding exponent for a particular axis
    return numerator/denominator_sum;



# TODO: compute the argmax of each gridpoint in one-hot form
# onehot_argmax = to_onehot(np.argmax(xy, axis=-1))


# plot the softmax
fig, axs = plt.subplots(2, 4, figsize=(20, 10))
fig.tight_layout()

# TODO: make the plots as specified on the sheet (nicest is in a grid which you

for i in range (len(lamb_array)):
    lamb = lamb_array[i]

    sfm1 = softmax(sigmas,0,lamb = lamb)
    sfm2 = softmax(sigmas,1,lamb = lamb)

    im = axs[0,i].imshow(sfm1)
    axs[0,i].set_title(r'SoftMax for $\lambda = $ ' + str(lamb))
    plt.colorbar(im,ax = axs[0,i],fraction=0.046)
    im2 = axs[1,i].imshow(sfm1)
    axs[1,i].set_title(r'SoftMax for $\lambda = $ ' + str(lamb))
    plt.colorbar(im2,ax = axs[1,i],fraction=0.046)


# plot the onehot argmax

##creating 2d vector like meshgrid it has a shape 2*100*100
a,b = sigmas[0].shape
```

```
onehot =  np.zeros(len(sigmas)*sigmas[0].size).reshape(2,a,b)

index1 = np.where(sig1<sig2)
index2 = np.where(sig1>=sig2)

onehot[0][index1] = 1
onehot[1][index2] = 1

im = axs[0,3].imshow(onehot[0])
axs[0,3].set_title(r'OneShot first component')
plt.colorbar(im,ax = axs[0,3],fraction=0.046)
im2 = axs[1,3].imshow(onehot[1])
axs[1,3].set_title(r'OneShot second component')
plt.colorbar(im2,ax = axs[1,3],fraction=0.046)
```

Out[ ]: <matplotlib.colorbar.Colorbar at 0x7fe713be5ea0>



In [ ]:
```
Image(filename='2.4.jpg')
```

(e). Bonus $\lim_{\nu \to \infty} lse(\vec{\sigma}; \nu) = max(\vec{\sigma})$.

$$lse(\vec{\sigma}, \nu) = \frac{1}{\nu} \log\left(\sum_{j=0}^{K} \exp(\nu \sigma_j^\circ) \right)$$

Clearly as $\nu \to \infty$, $\lim_{\nu \to \infty} lse$ is undefined

thus we can use l'Hôpital's rule

and focus on limit (derivative of numerator and denominator

$$= \frac{\sum \sigma_j^\circ \exp(\nu \sigma_j^\circ)}{\sum_{j=0}^{K} \exp(\nu \sigma_j^\circ)}$$

Clearly we see major contribution we get from the term with largest σ_j°.

thus $\lim_{\nu \to \infty} lse = max(\vec{\sigma})$

3. Linear regions of MLPs

(a)

(a).1

In [ ]:
```
class Abs(nn.Module):
    """Absolute value activation function. You can experiment with this insteac
    def forward(self, x):
        return x.abs()

# define NN architecture.
class MLPShallow(nn.Module):
    def __init__(self,n,p,k):
        """
        Attributes:
        n : no. of inputs (dimensions)
        p : no. of neurons in hidden layer
        k : no. of nuerons in output layer
        """
        super(MLPShallow,self).__init__()
        # TODO: initialize Linear Layers and the activation as specified on the
        self.hidden = nn.Linear(n, p, bias=True)     # hidden layer
        self.relu = nn.ReLU()                        # relu activation function
        self.abs = Abs()                             # absolute activation funct
        self.output = nn.Linear(p, k, bias=True)     # output layer


    def forward(self, x):
        """
        Attributes:
        x : input matrix
        """
        # TODO: pass the input x through the layers and return the output
        a = self.relu(self.hidden(x))   # input -> hidden       #a1 = ReLU(W_0
        # a = self.abs(self.hidden(x))    # input -> hidden       #a1 = Abs(W_(
        y = self.output(a)              # hidden -> output      #y = W_1 a_1 +

        return y
```

(a).2 How many paramters does the model have?

Given equations,

$$
\begin{aligned}
a_0 &= \mathbf{x} \\
a_{i+1} &= ReLU(\mathbf{W}_i \mathbf{a}_i + \mathbf{b}_i) \; for \; i \in 0, \ldots, H-1 \\
\mathbf{y} &= \mathbf{W}_H \mathbf{a}_H + \mathbf{b}_H
\end{aligned}
$$

Now, when done dimension analysis on H=1, we can write (after taking transpose):

$$\underset{m\times p}{\mathbf{a}_1} = ReLU(\underset{m\times n}{\mathbf{x}}\ \underset{n\times p}{\mathbf{W}_0} + \underset{1\times p}{\mathbf{b}_0})$$

$$\underset{m\times k}{\mathbf{y}} = \underset{m\times p}{\mathbf{a}_1}\ \underset{p\times k}{\mathbf{W}_1} + \underset{1\times k}{\mathbf{b}_1}$$

where,

- $m =$ no. of observations,
- $n =$ size of input layer,
- $p =$ size of hidden layer, \&
- $k =$ size of output layer

So, for 2 dimensional input on 20 neuron hidden layer with single neuron ouput layer, we get, $n = 2,\ p = 20,\ k = 1$,

Therefore,

- Parameters in Hidden layer, $P_{hidden} = n \times p + 1 \times p = 40 + 20 = 60$
- Parameters in Output layer, $P_{output} : p \times k + 1 \times k = 20 + 1 = 21$

Hence,

Total Parameters in the model, $P_{hidden} + P_{output} = 81$

In [ ]:
```
# the above answer can also be proved by calculating the no. of params in the m
from torchsummary import summary
_ = summary(MLPShallow(2,20,1))
```

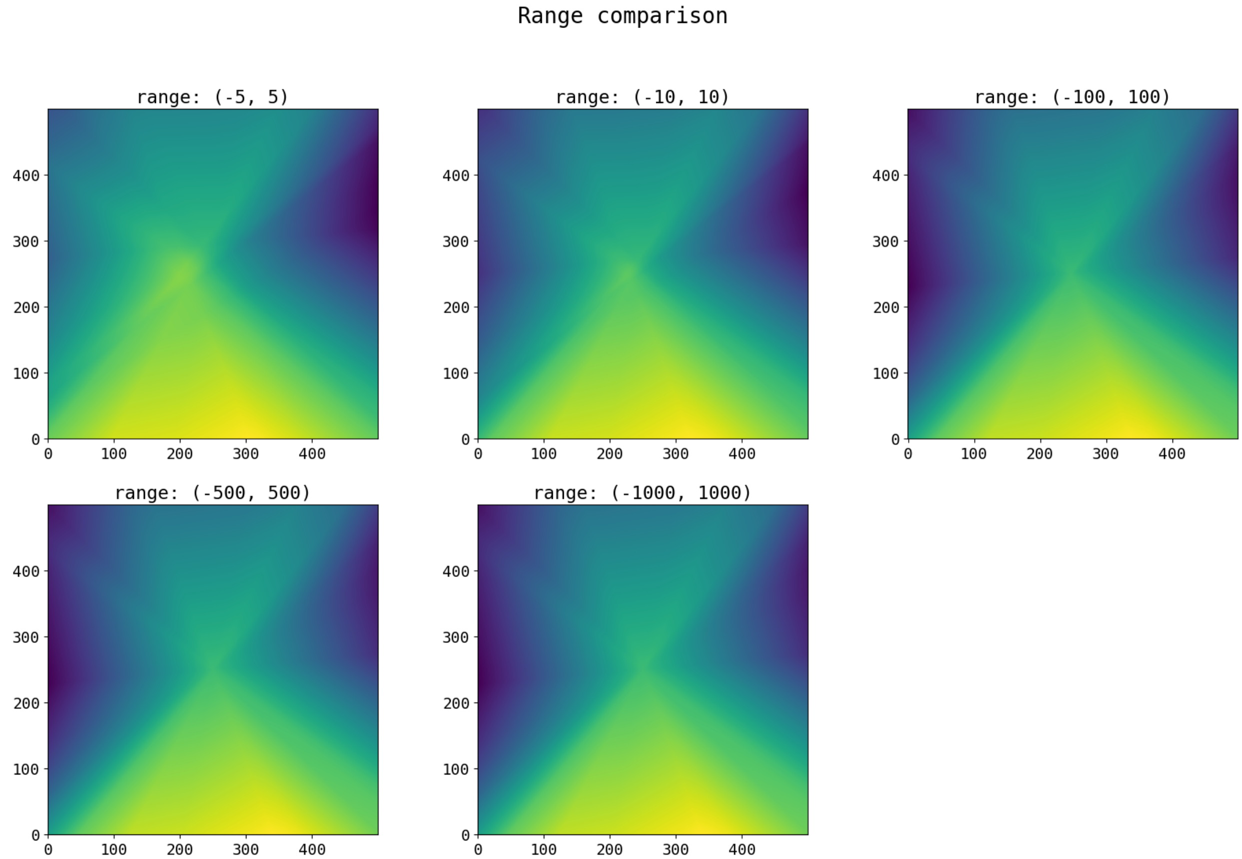
```
=================================================================
Layer (type:depth-idx)                   Param #
=================================================================
├─Linear: 1-1                            60
├─ReLU: 1-2                              --
├─Abs: 1-3                               --
├─Linear: 1-4                            21
=================================================================
Total params: 81
Trainable params: 81
Non-trainable params: 0
=================================================================
```

- as this also shows, the total parameters are 81.

(b)

In [ ]:
```
def visualize_model(model, res=500, bound=5):
    # TODO: implement a function that takes the model (the MLP), and builds a
    #       grid of points in [-bound, bound] x [-bound, bound], passes them
    #       through the model and returns the result in the shape of an image

    #grids of points
    x1 = np.linspace(-bound,bound,res)
    x2 = np.linspace(-bound,bound,res)
    x1v,x2v = np.meshgrid(x1,x2)
```

```
    # genrating the input torch.tensor
    x = np.stack((x1v.ravel(),x2v.ravel()),axis=1)
    x = torch.from_numpy(x).to(torch.float32)

    # passing through model
    y = model(x)

    return y.detach().numpy().reshape(res,res)      #reshaping the output vecto



# TODO: instantiate the model and make the visualizations as requested in the t
# NOTE: If you get a constant output, you got an unlucky initialization. Simply

model = MLPShallow(2,20,1)
bounds = [5,10,100,500,1000]
imgs = [visualize_model(model,res=500, bound=bound) for bound in bounds]

# plotting the model outputs for different ranges

plt.figure(figsize=(20,12))
plt.suptitle('Range comparison', fontsize=20, y=0.99)
plt.tight_layout()
#setting no. of rows and columns for subplot
ncols = 3
nrows = len(imgs) // ncols + (len(imgs) % ncols > 0) # calculating number of ro

for n,im in enumerate(imgs):
    #adding subplot iteratively
    ax = plt.subplot(nrows, ncols, n + 1)

    ax.imshow(im,origin='lower')
    ax.set_title(f'range: {-bounds[n],bounds[n]}')
    ax.set_aspect('equal')
```

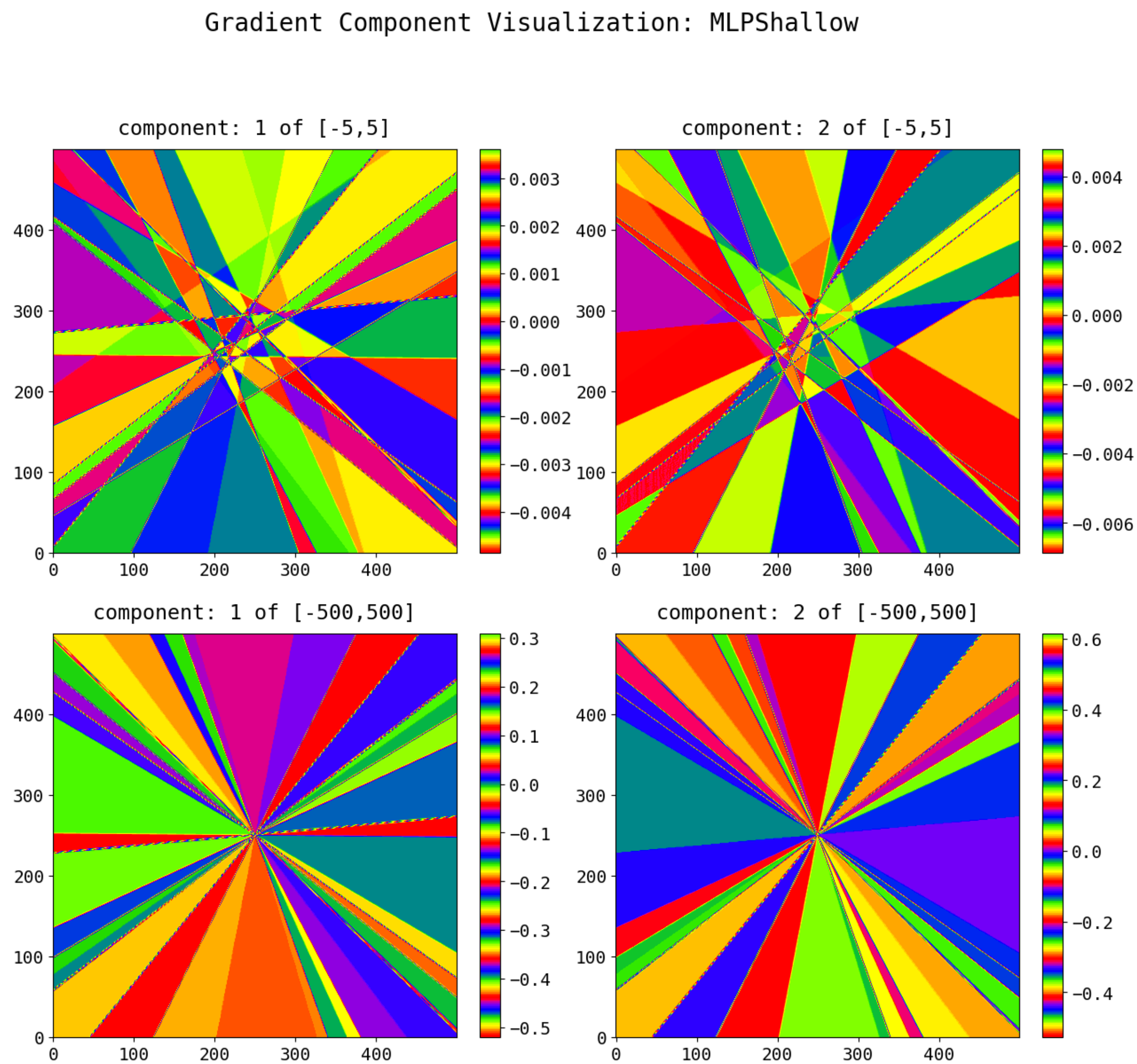


- we can observe that as the range increases, the structure gets clearer and more refined.
- after range: [-100,100], the structure doesn't improve by much factor as compared between range: [-10,10] to rangg: [-100,100]
- a possible reason for this could be that in the samll range, because of the fixed no. of points, the adjacent points are not sparse and therefore are quite close to each other, hence kind of blurred.
- as the range increaes, the sparsity of the points increases and they are able to map much larger spaces, hence much more clearer strcuture.

(c)

In [ ]:
```
# TODO: compute the spatial gradient of the network outputs (as an image) from
#       using np.gradient, and visualize using matplotlib's prism colormap

c1, c2 = np.gradient(imgs[0]) #for range (-5,5)
c3, c4 = np.gradient(imgs[3]) #for range (-500,500)

# plotting the model outputs for different ranges

fig = plt.figure(figsize=(15,13))
plt.suptitle('Gradient Component Visualization: MLPShallow', fontsize=20, y=1)
plt.tight_layout()

#setting no. of rows and columns for subplot
ncols = 2
nrows = 2
```

```
c = [c1,c2,c3,c4]
label = ['component: 1 of [-5,5]','component: 2 of [-5,5]','component: 1 of [-5

for n,(ci,li) in enumerate(zip(c,label)):
    #adding subplot iteratively
    ax = plt.subplot(nrows, ncols, n + 1)

    cim = ax.imshow(ci,origin='lower',cmap='prism')
    ax.set_title(li, y=1.02)
    ax.set_aspect('equal')
    fig.colorbar(cim, ax=ax, fraction=0.046)
```



Inference:

- `np.gradient` gives us the gradient of the input array with respect to each dimension with the same shape
- the pattern in the component images of respective ranges is almost similar with a few changes but the level changes which is represented by different colors

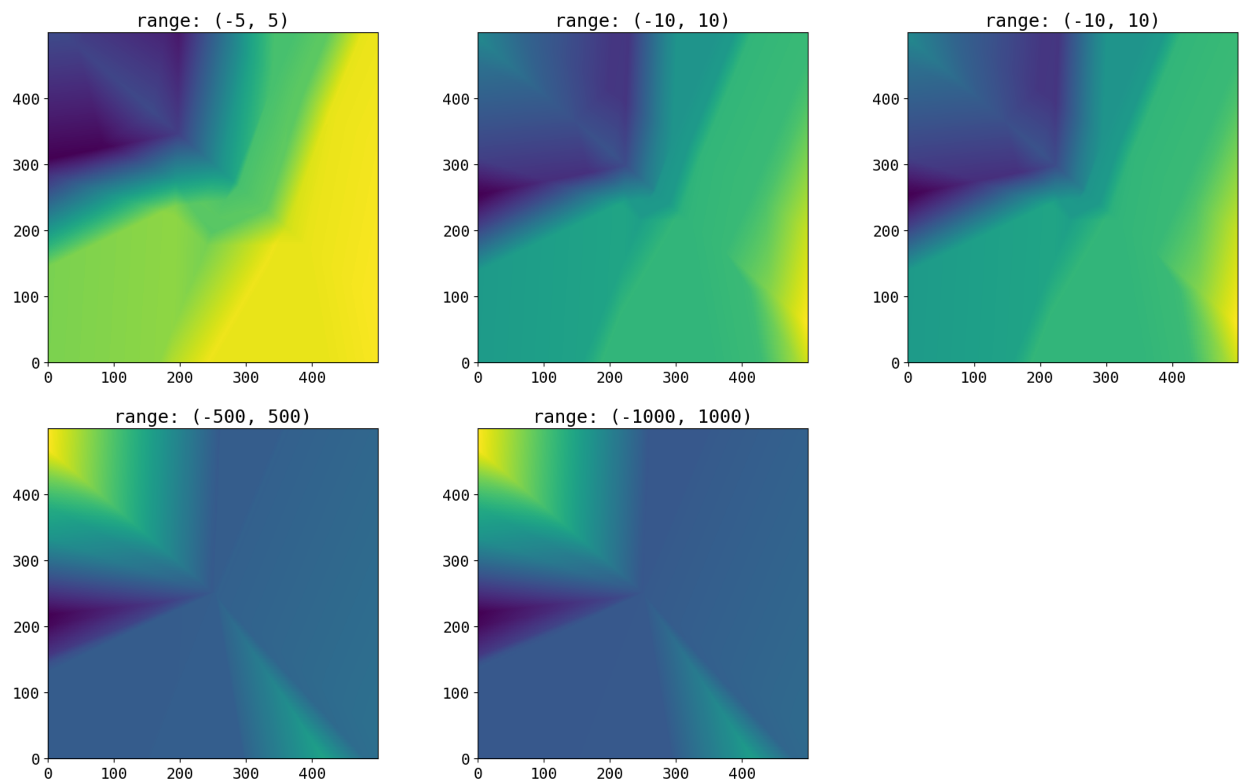
- for the range [-5,5], pattern is almost chaotic but for range [-500,500], the structure is quite clear
- also, the colorbar of component 2 of both the levels are quite spread out which indicates the the levels are spread out too as compared to the component 1

(d)

In [ ]:
```
# define NN architecture.
class MLPDeep(nn.Module):
    def __init__(self,n=2,p=5,k=1):
        """
        Attributes:
        n : no. of inputs (dimensions)
        p : no. of neurons in hidden layers
        k : no. of nuerons in output layer
        """
        super(MLPDeep,self).__init__()
        # TODO: initialize Linear Layers and the activation as specified on the
        self.multi_hidden = nn.Sequential(
                            nn.Linear(n, p),        # hidden layer 1
                            nn.ReLU(),                # relu
                            nn.Linear(p, p),        # hidden layer 2
                            nn.ReLU(),
                            nn.Linear(p, p),        # hidden layer 3
                            nn.ReLU(),
                            nn.Linear(p, p),        # hidden layer 4
                            nn.ReLU(),
                            nn.Linear(p,k))          # output layer


    def forward(self, x):
        # TODO: pass the input x through the layers and return the output
        """
        Attributes:
        x : input matrix
        """
        # TODO: pass the input x through the layers and return the outpu
        y = self.multi_hidden(x)
        return y

# TODO: repeat the visualizations from above
model_deep = MLPDeep(2,5,1)
bounds = [5,10,10,500,1000]
img = [visualize_model(model_deep,res=500, bound=bound) for bound in bounds]

# plotting the model outputs for different ranges

plt.figure(figsize=(20,12))
plt.suptitle('Deep Learning Model: Range comparison', fontsize=20, y=0.99)
plt.tight_layout()
#setting no. of rows and columns for subplot
ncols = 3
nrows = len(img) // ncols + (len(img) % ncols > 0) # calculating number of rows

for n,im in enumerate(img):
    #adding subplot iteratively
    ax = plt.subplot(nrows, ncols, n + 1)
```

```
    ax.imshow(im,origin='lower')
    ax.set_title(f'range: {-bounds[n],bounds[n]}')
    ax.set_aspect('equal')
```

Deep Learning Model: Range comparison



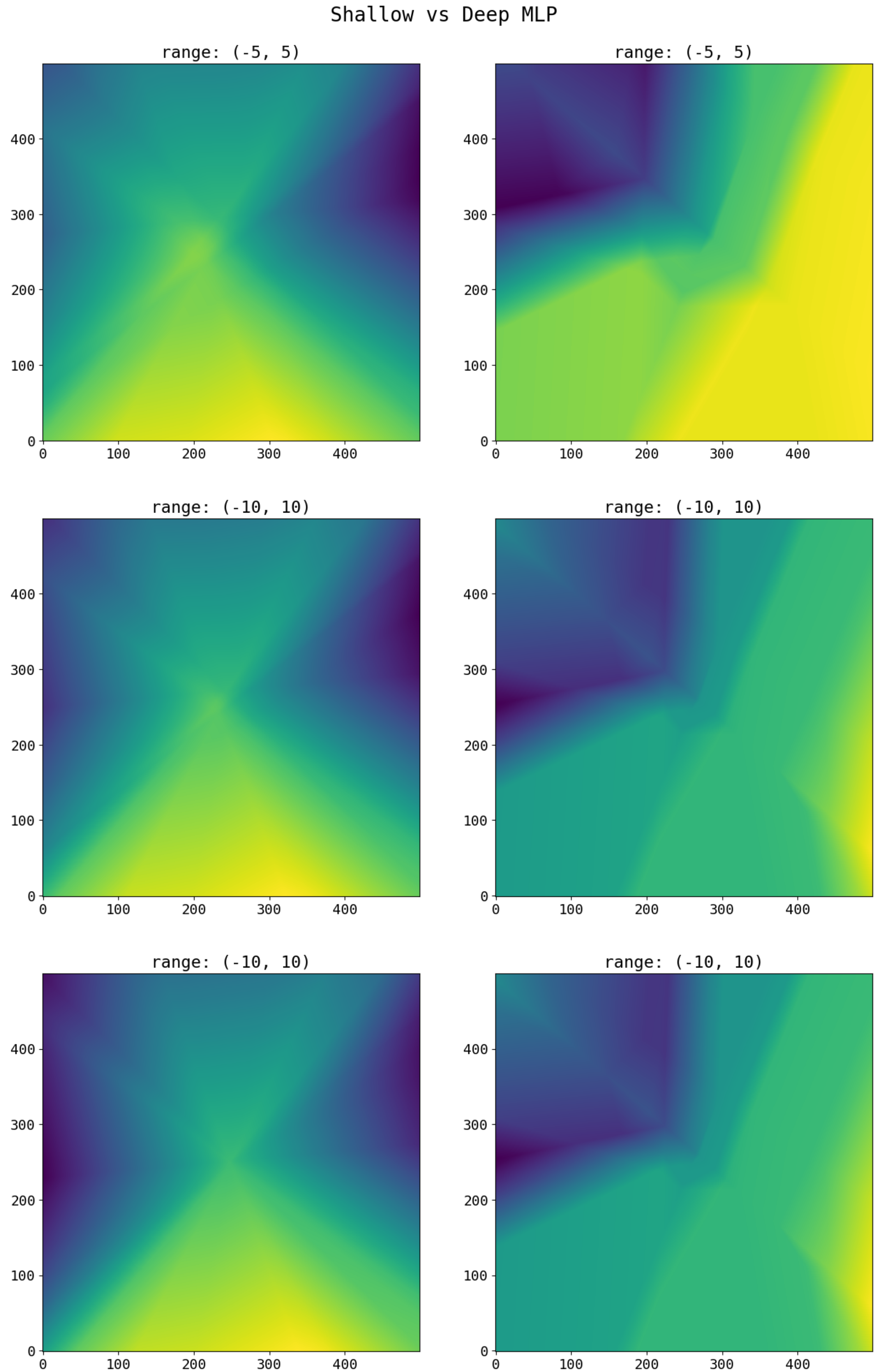
Inference:

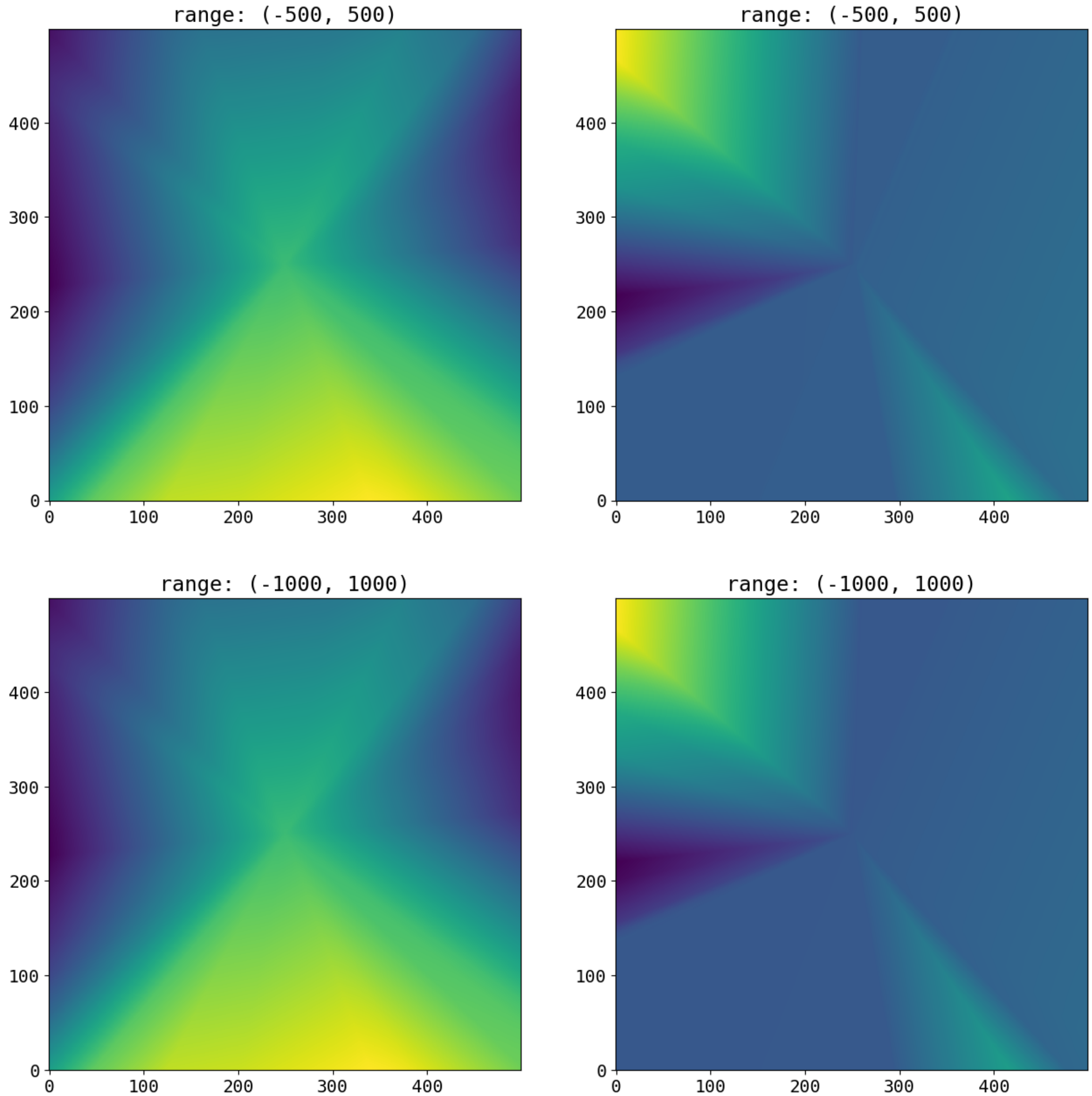
- The results are actually impressive!
- Structures are quite clear even from the beginning, although their shape changes as we increase the range (we are not sure why)
- Ones with the lower range are more rough or would say that it has more distinctions in the structures whereas the range of [-1000,1000] looks quite different and has more smoother surface.

In [ ]:

```
#combining the img arrays alternatively for comparison plotting
img_com = [x for y in zip(imgs,img) for x in y]
bnd_com = [x for y in zip(bounds,bounds) for x in y]

plt.figure(figsize=(15,40))
plt.suptitle('Shallow vs Deep MLP', fontsize=20, y=0.9)
plt.tight_layout()
#setting no. of rows and columns for subplot
ncols = 2
nrows = len(img_com) // ncols + (len(img_com) % ncols > 0) # calculating numbe

for n, im_com in enumerate(img_com):
    #adding subplot iteratively
    ax = plt.subplot(nrows, ncols, n + 1)

    ax.imshow(im_com,origin='lower')
```

```
    ax.set_title(f'range: {-bnd_com[n],bnd_com[n]}')
    ax.set_aspect('equal')
```

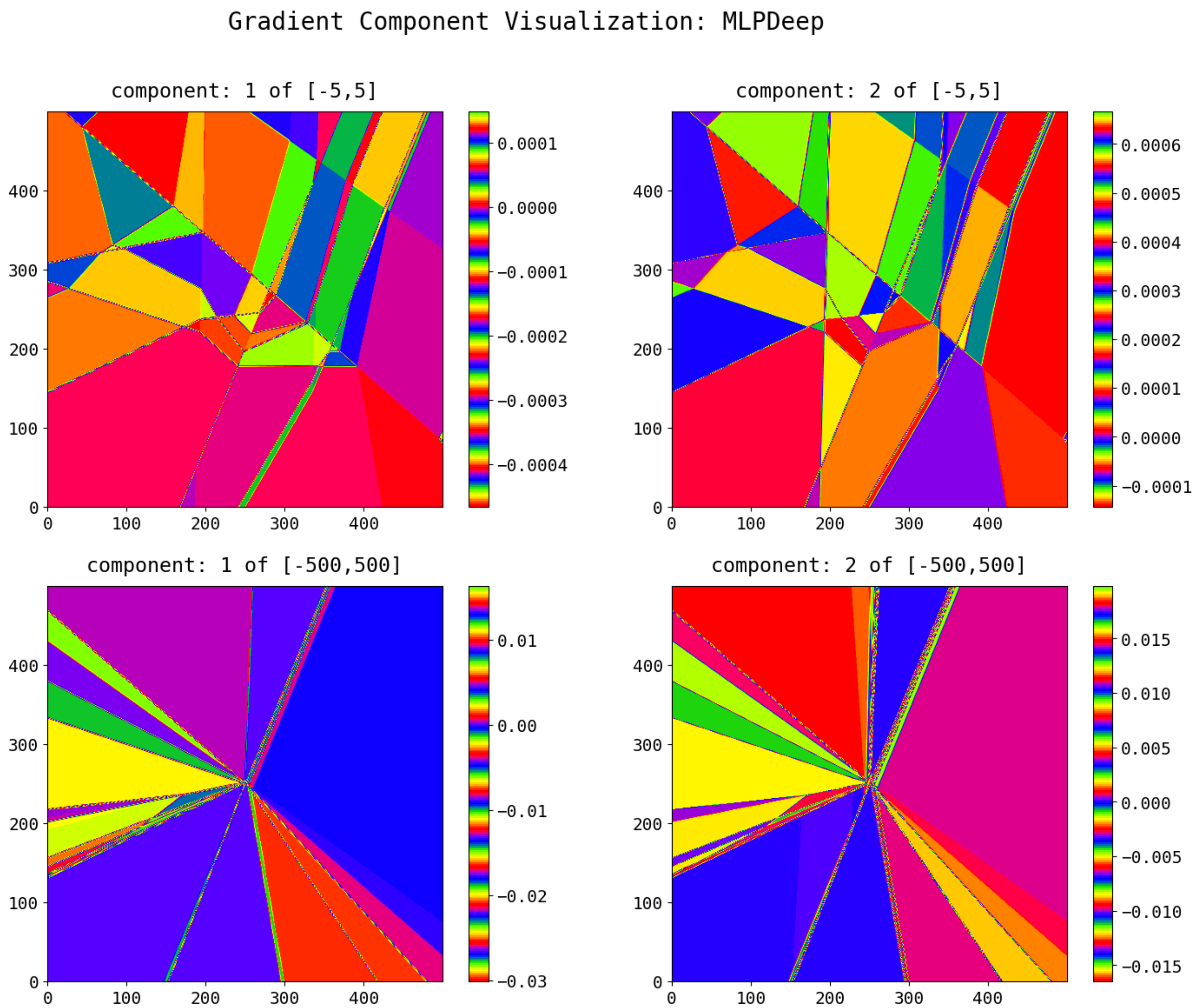

Shallow vs Deep MLP
range: (-5, 5)
range: (-5, 5)
range: (-10, 10)
range: (-10, 10)
range: (-10, 10)
range: (-10, 10)
0
100
200
300
400



In [ ]:

```
# TODO: compute the spatial gradient of the network outputs (as an image) from
#       using np.gradient, and visualize using matplotlib's prism colormap

c1, c2 = np.gradient(img[0]) #for range (-5,5)
c3, c4 = np.gradient(img[3]) #for range (-500,500)

# plotting the model outputs for different ranges

fig = plt.figure(figsize=(17,13))
plt.suptitle('Gradient Component Visualization: MLPDeep', fontsize=20, y=0.97)
plt.tight_layout()

#setting no. of rows and columns for subplot
ncols = 2
nrows = 2

c = [c1,c2,c3,c4]
label = ['component: 1 of [-5,5]','component: 2 of [-5,5]','component: 1 of [-5
```

```python
for n,(ci,li) in enumerate(zip(c,label)):
    #adding subplot iteratively
    ax = plt.subplot(nrows, ncols, n + 1)

    cim = ax.imshow(ci,origin='lower',cmap='prism')
    ax.set_title(li, y=1.02)
    ax.set_aspect('equal')
    fig.colorbar(cim, ax=ax, fraction=0.046)
```



Inference:

- here also, the pattern is similiar but has some little differences
- although color map ranges doesn't follow the usual behaviour now