sheet05_num3-4

November 24, 2024

1 Sheet 5

[11]:
```
import os
import pandas as pd
```

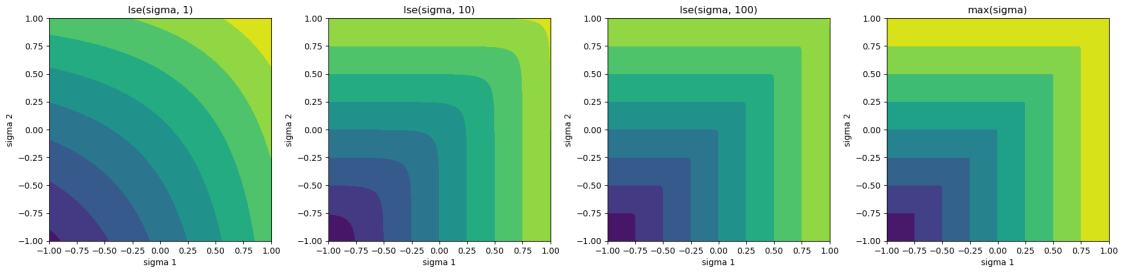
1.1 3 Log-sum-exp and soft(arg)max

1.1.1 (b)

[123]:
```
import numpy as np
from matplotlib import pyplot as plt

def softmax(sigma, lam):
    denom = sum([np.exp(lam*s) for s in sigma])
    return [np.exp(lam*s) / denom for s in sigma]

def lse(sigma, lam):
    return np.log(sum([np.exp(lam*s) for s in sigma])) / lam


# (a)
print("(a)")
print(softmax([1, 2, 3], 1))
print(softmax([11, 12, 13], 1))
print(softmax([10, 20, 30], 1))


fig, ax = plt.subplots(1, 4)
s1_all = np.linspace(-1, 1, 100)
s2_all = np.linspace(-1, 1, 100)
for i, lam in enumerate([1, 10, 100]):
    s1_grid, s2_grid = np.meshgrid(s1_all, s2_all)
    lse_val = np.array([[lse([s1, s2], lam) for s1 in s1_all] for s2 in s2_all])

    cs = ax[i].contourf(s1_grid, s2_grid, lse_val)
    ax[i].set_xlabel('sigma 1')
    ax[i].set_ylabel('sigma 2')
    ax[i].set_title(f'lse(sigma, {lam})')
```

```
    ax[i].set_aspect('equal', adjustable='box')

lse_val = np.array([[max(s1, s2) for s1 in s1_all] for s2 in s2_all])
ax[3].contourf(s1_grid, s2_grid, lse_val)
ax[3].set_xlabel('sigma 1')
ax[3].set_ylabel('sigma 2')
ax[3].set_title('max(sigma)')

ax[3].set_aspect('equal', adjustable='box')


fig.set_figwidth(23)
plt.show()
```

```
(a)
[0.09003057317038046, 0.24472847105479767, 0.6652409557748219]
[0.09003057317038046, 0.24472847105479764, 0.6652409557748219]
[2.061060046209062e-09, 4.539786860886666e-05, 0.9999546000703311]
```

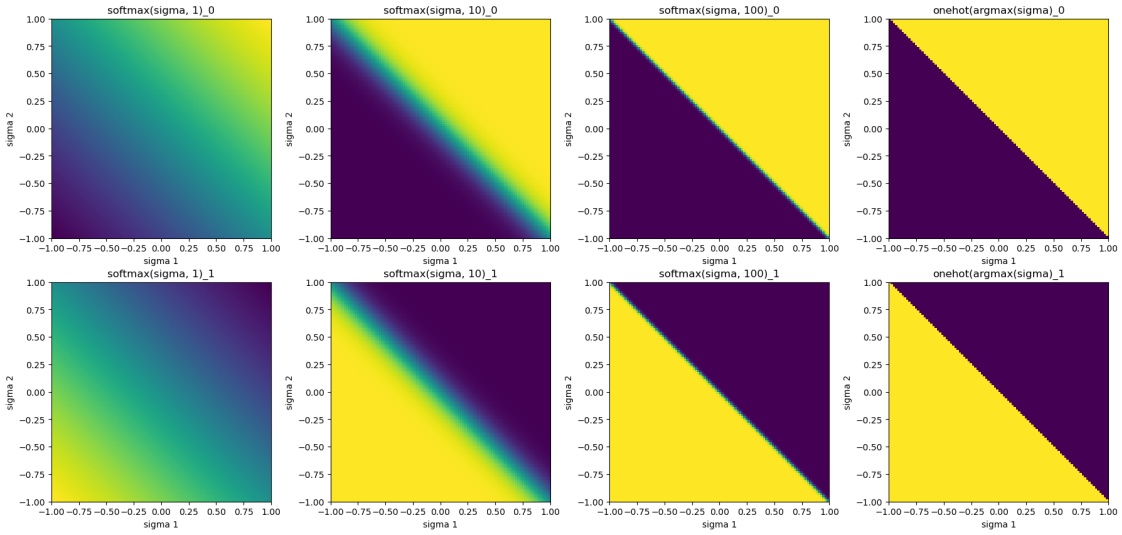


The bigger the lambda, the more the lse plot looks like the max(sigma) plot. Smaller lamdas make the plot more rounded.

1.1.2 (c)

[121]:
```
fig, ax = plt.subplots(2, 4)
for i, lam in enumerate([1, 10, 100]):
    for j in [0, 1]:
        softmax_val = np.array([[softmax([s1, s2], lam)[j] for s1 in s1_all]␣
 ↪for s2 in s2_all])
        ax[j, i].imshow(softmax_val, extent=[-1, 1, -1, 1])
        ax[j, i].set_xlabel('sigma 1')
        ax[j, i].set_ylabel('sigma 2')
        ax[j, i].set_title(f'softmax(sigma, {lam})_{j}')
        ax[j, i].set_aspect('equal', adjustable='box')

for j in [0, 1]:
```

```
    onehot_val = np.array([[j if s1 <= s2 else 1 - j for s1 in s1_all] for s2␣
 ↪in s2_all])
    ax[j, 3].imshow(onehot_val, extent=[-1, 1, -1, 1])
    ax[j, 3].set_xlabel('sigma 1')
    ax[j, 3].set_ylabel('sigma 2')
    ax[j, 3].set_title(f'onehot(argmax(sigma)_{j}')
    ax[j, 3].set_aspect('equal', adjustable='box')


fig.set_figwidth(22)
fig.set_figheight(10)
```



The bigger the lambda, the more the softmax plot looks like the onehot(argmax) plot. Smaller lamdas make the edge from 0 to 1 more blurry.

P.S For solutions of the other parts of this problem see hand-written notes.

1.2 4 Linear regions of MLPs

[ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

1.2.1 (a)

[105]:
```
class ShallowNet(nn.Module):
    def __init__(self):
        super(ShallowNet, self).__init__()
        self.fc1 = nn.Linear(2, 20)
        self.fc2 = nn.Linear(20, 1)
```

```
    # x represents our data
    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)

        x = self.fc2(x)
        return x

shallow_nn = ShallowNet()
print(shallow_nn)
```

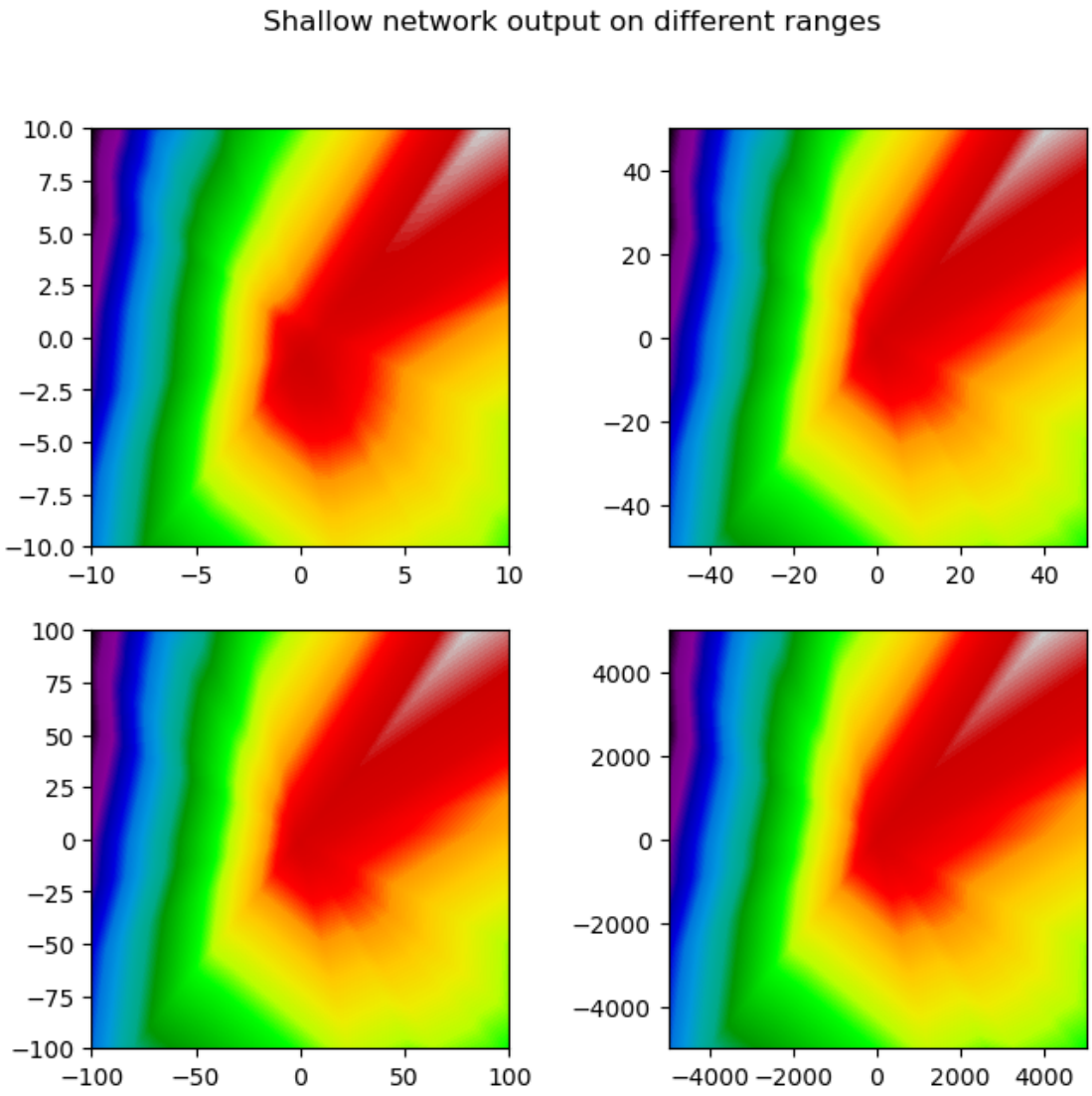
```
ShallowNet(
  (fc1): Linear(in_features=2, out_features=20, bias=True)
  (fc2): Linear(in_features=20, out_features=1, bias=True)
)
```

The model has 20 weights + 1 bias on linear layers. This model has 2 linear layers, so the answer is 21*2=42 parameters.

1.2.2 (b)

[108]:
```
ranges = [(-10, 10), (-50, 50), (-100, 100), (-5000, 5000)]

shallow_vals = []

for i, x_range in enumerate(ranges):
    x1_all = np.linspace(x_range[0], x_range[1], 500)
    x2_all = np.linspace(x_range[0], x_range[1], 500)

    val = np.array([[shallow_nn(torch.tensor([x1, x2], dtype=torch.float32)).
 ↪item() for x1 in x1_all] for x2 in x2_all])
    shallow_vals.append(val)
```

[113]:
```
fig, ax = plt.subplots(len(ranges) // 2, 2)


for i, x_range in enumerate(ranges):
    ax[i // 2, i % 2].imshow(shallow_vals[i], cmap=mpl.
 ↪colormaps['nipy_spectral'],  extent=[x_range[0], x_range[1], x_range[0],␣
 ↪x_range[1]])

fig.suptitle("Shallow network output on different ranges")

fig.set_figheight(7)
fig.set_figwidth(8)
plt.show()
```



We do not have to zoom out too much to see the whole structure, the range [-50, 50] is already enough.

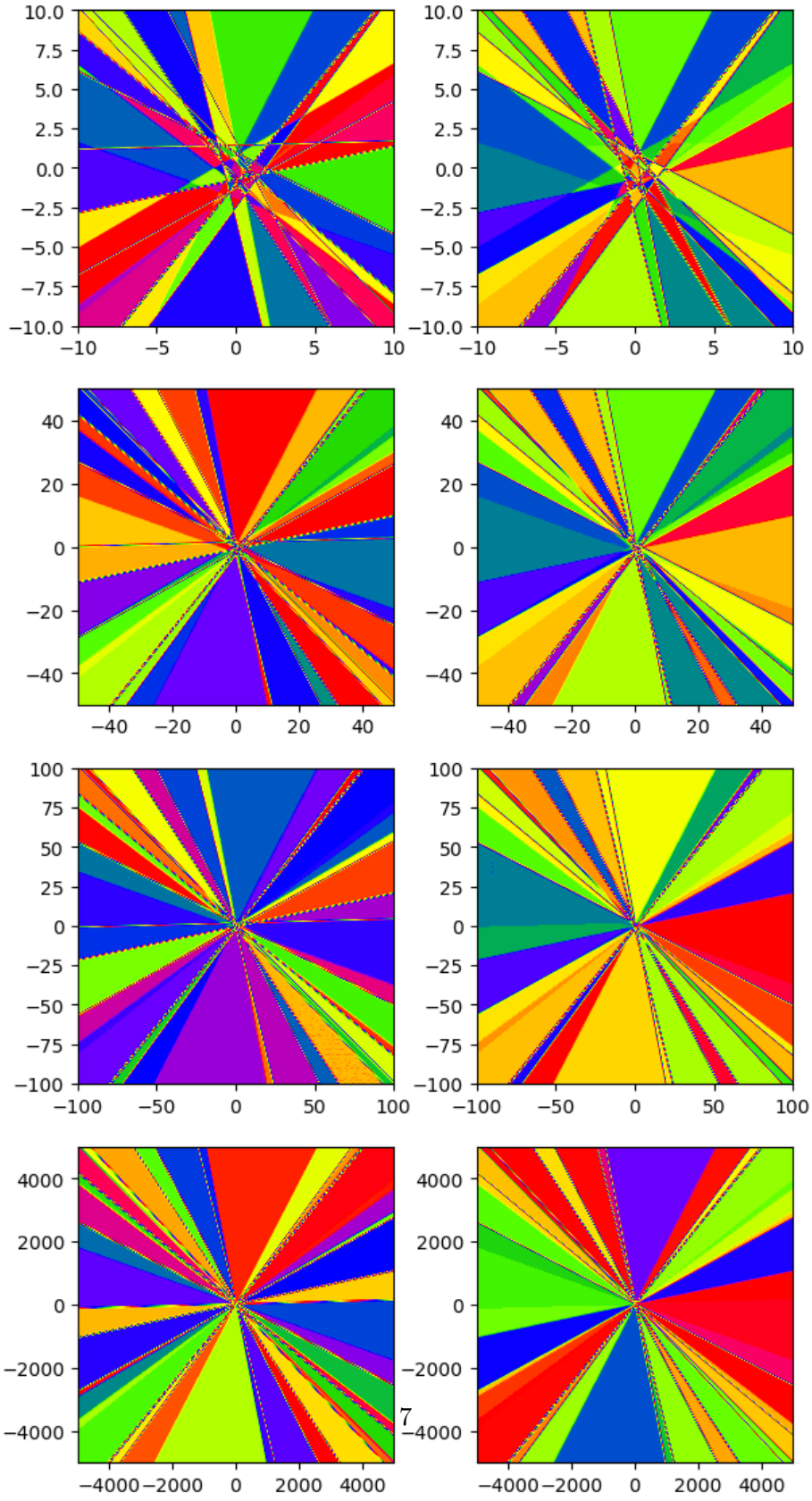
1.2.3 (c)

[ ]:
```
import matplotlib as mpl
```

[119]:
```
fig, ax = plt.subplots(len(ranges), 2)
for i, x_range in enumerate(ranges):
    shallow_gradient = np.gradient(shallow_vals[i])
    for j in range(2):
        ax[i, j].imshow(shallow_gradient[j], cmap=mpl.colormaps['prism'],␣
↪extent=[x_range[0], x_range[1], x_range[0], x_range[1]])
```

```
fig.suptitle("Gradient of the shallow network output on different ranges")

fig.set_figwidth(7)
fig.set_figheight(14)
plt.show()
```

Gradient of the shallow network output on different ranges



The value of the gradients can be described as intersecting lines. They do not intersect in zero, as it might have seemed, if we would have looked at the plots on the big range. Although all their intersections are in the [-3, 3] range. The number of these lines probably correlates with the number of parameters in the model.

1.2.4 (d)

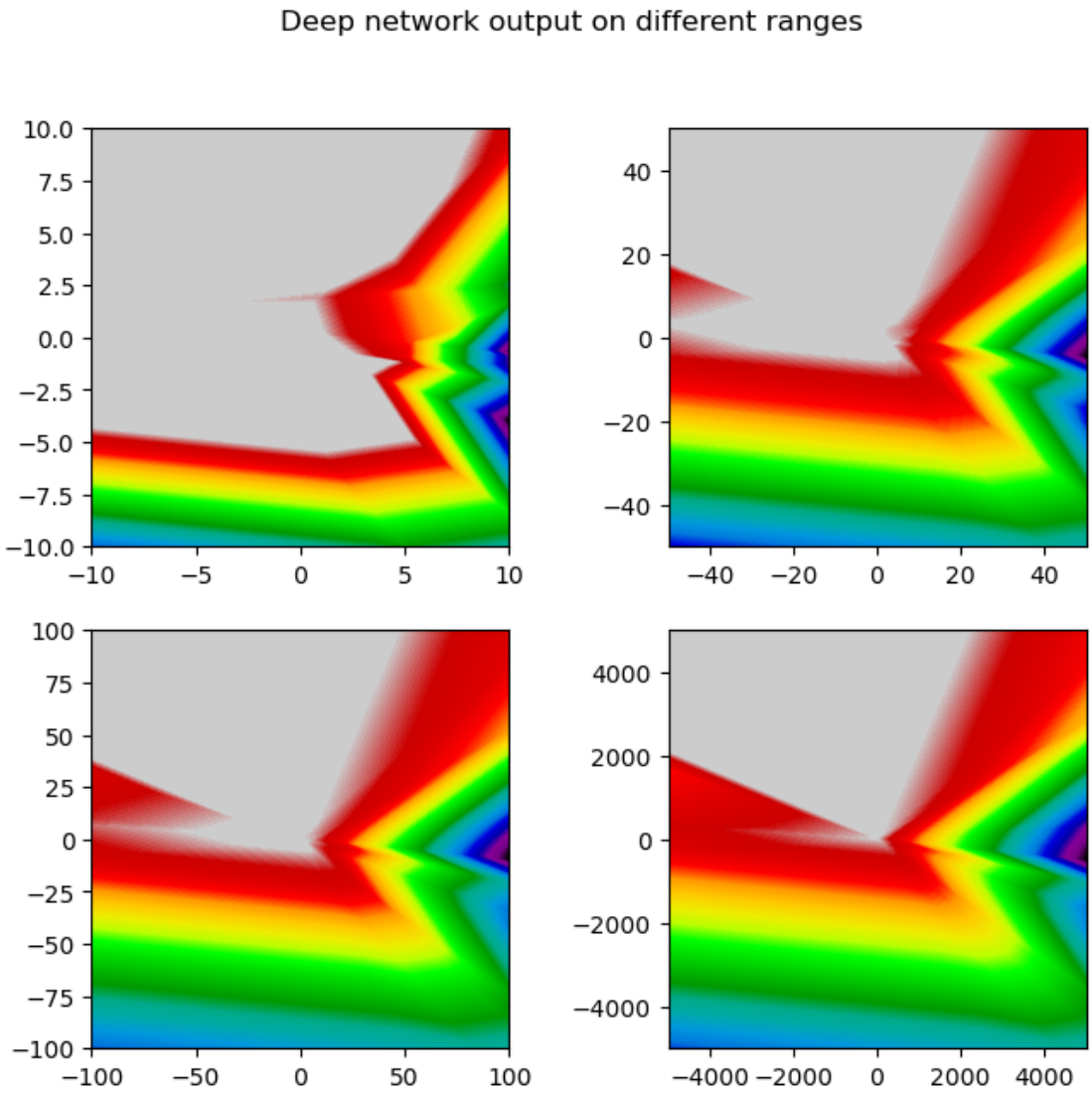
[84]:
```
class DeepNet(nn.Module):
    def __init__(self):
        super(DeepNet, self).__init__()
        self.fc1 = nn.Linear(2, 5)
        self.fc2 = nn.Linear(5, 5)
        self.fc3 = nn.Linear(5, 5)
        self.fc4 = nn.Linear(5, 5)
        self.fc5 = nn.Linear(5, 1)

    # x represents our data
    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)
        x = F.relu(x)

        x = self.fc5(x)
        return x

deep_nn = DeepNet()
print(deep_nn)
```

```
DeepNet(
  (fc1): Linear(in_features=2, out_features=5, bias=True)
  (fc2): Linear(in_features=5, out_features=5, bias=True)
  (fc3): Linear(in_features=5, out_features=5, bias=True)
  (fc4): Linear(in_features=5, out_features=5, bias=True)
  (fc5): Linear(in_features=5, out_features=1, bias=True)
)
```

[115]:
```
deep_vals = []

for i, x_range in enumerate(ranges):
    x1_all = np.linspace(x_range[0], x_range[1], 500)
```

```
    x2_all = np.linspace(x_range[0], x_range[1], 500)

    val = np.array([[deep_nn(torch.tensor([x1, x2], dtype=torch.float32)).
 ↪item() for x1 in x1_all] for x2 in x2_all])
    deep_vals.append(val)
```

[117]:
```
fig, ax = plt.subplots(len(ranges) // 2, 2)


for i, x_range in enumerate(ranges):
    ax[i // 2, i % 2].imshow(deep_vals[i], cmap=mpl.colormaps['nipy_spectral'],␣
 ↪extent=[x_range[0], x_range[1], x_range[0], x_range[1]])

fig.suptitle("Deep network output on different ranges")

fig.set_figheight(7)
fig.set_figwidth(8)
plt.show()
```

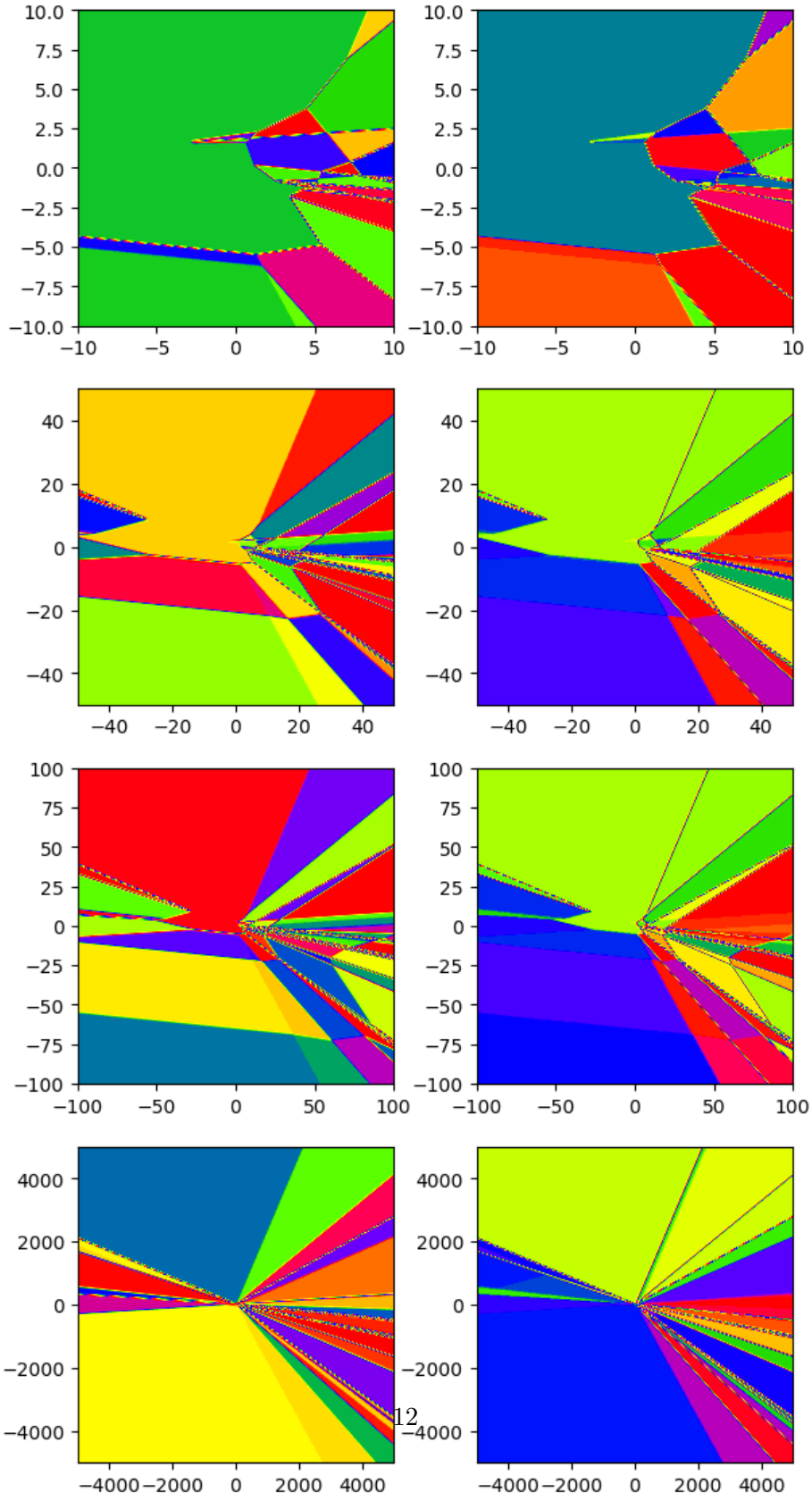


The shapes that are formed by the deep network are much less trivial. The image changes more drastically while zooming out. But it estimates parameters approximately twice as long as the shallow network, while having approximately the same number of parameters.

[118]:
```
fig, ax = plt.subplots(len(ranges), 2)
for i, x_range in enumerate(ranges):
    deep_gradient = np.gradient(deep_vals[i])
    for j in range(2):
        ax[i, j].imshow(deep_gradient[j], cmap=mpl.colormaps['prism'],␣
 ↪extent=[x_range[0], x_range[1], x_range[0], x_range[1]])

fig.suptitle("Gradient of the deep network output on different ranges")
```

```
fig.set_figwidth(7)
fig.set_figheight(14)
plt.show()
```

Gradient of the deep network output on different ranges



The shape of the gradients is also more complicated, although on the big scale they still look like lines intersecting in the middle (which they are not).