CodingLab7

June 19, 2022

Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2022

Name: Moritz Kniebel, Vanessa Tsingunidis

1 Coding Lab 7

[71]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import scipy.optimize as opt
import scipy.io as io
import scipy as sp

mpl.rc("savefig", dpi=72)

import itertools

sns.set_style('whitegrid')
%matplotlib inline
```

1.1 Task 1: Fit RF on simulated data

We will start with toy data generated from an LNP model neuron to make sure everything works right. The model LNP neuron consists of one Gaussian linear filter, an exponential nonlinearity and a Poisson spike count generator. We look at it in discrete time with time bins of width δt. The model is:

$$c_t \sim Poisson(r_t) r_t = \exp(w^T s_t) \cdot \Delta t \cdot R$$

Here, c_t is the spike count in time window t of length Δt, s_t is the stimulus and w is the receptive field of the neuron. The receptive field variable `w` is 15×15 pixels and normalized to $||w|| = 1$. A stimulus frame is a 15 × 15 pixel image, for which we use uncorrelated checkerboard noise. R can be used to bring the firing rate into the right regime (e.g. by setting $R = 50$).

For computational ease, we reformat the stimulus and the receptive field in a 225 by 1 array. The function `sampleLNP` can be used to generate data from this model. It returns a spike count vector `c` with samples from the model (dimensions: 1 by nT = $T/\Delta t$), a stimulus matrix `s` (dimensions: 225 × nT) and the mean firing rate `r` (dimensions: nT × 1).

Here we assume that the receptive field influences the spike count instantaneously just as in the above equations. Implement a Maximum Likelihood approach to fit the receptive field.

To this end simplify and implement the log-likelihood function $L(w)$ and its gradient $\frac{L(w)}{dw}$ with respect to w (`logLikLnp`). The log-likelihood of the model is

$$L(w) = \log \prod_t \frac{r_t^{c_t}}{c_t!} \exp(-r_t).$$

Plot the true receptive field, a stimulus frame, the spike counts and the estimated receptive field.

1.1.1 Calculations

Simplify the log likelihood analytically and derive the analytical solution for the gradient. (2 pts)

See also: How to use Latex in Jupyter notebook.

$$L(w) = \log \prod_t \frac{r_t^{c_t}}{c_t!} \exp(-r_t) = \sum_t \log \frac{r_t^{c_t}}{c_t!} \exp(-r_t) = \sum_t \log r_t^{c_t} - \log c_t! + \log \exp(-r_t) = \sum_t c_t \log r_t - \log c_t! - r_t$$

Because $c_t!$ does not depend on w, we can move it to an additive constant. Using $r_t = \exp(w^T s_t) dtR$ we obtain:

$$L(w) = \sum_t c_t (w^T s_t + dtR) - \exp(w^T s_t) dtR + const_1. = \sum_t c_t w^T s_t - \exp(w^T s_t) dtR + const_2.$$

Note that s_t denotes a vector and c_t a scalar, in slight abuse of notation.

For the gradient:

$$dL(w)/dw = \sum_t c_t s_t - s_t \exp(w^T s_t) dtR = \sum_t (c_t - \exp(w^T s_t) dtR) s_t$$

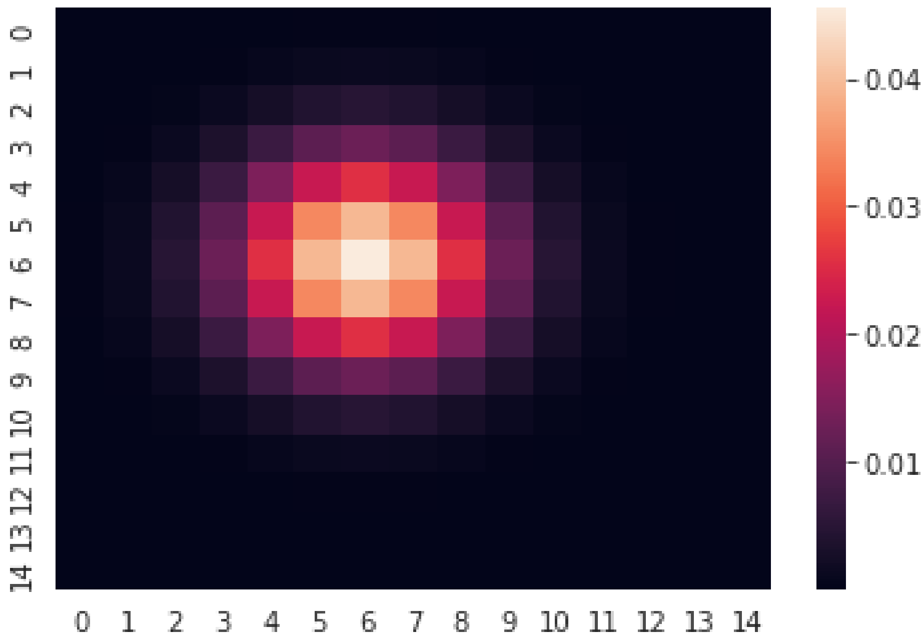
This is interesting and makes intuitive sense: for the gradient, each stimulus frame is weighted by the difference between the observed and predicted count.

1.1.2 Generate data

[72]:
```
def gen_gauss_rf(D, width, center=(0,0)):
    # Generate gaussian blop (image)
```

```
    sz = (D-1)/2
    x, y = sp.mgrid[-sz: sz + 1, -sz: sz + 1]
    x = x + center[0]
    y = y + center[1]
    w = np.exp(- (x ** 2/width + y ** 2 / width))
    w = w / np.sum(w.flatten())

    return w

w = gen_gauss_rf(15, 7, (1,1))
sns.heatmap(w)
```

[72]: <AxesSubplot:>



[73]:
```
def sample_lnp(w, nT, dt, R, v):
    '''Generate samples from an instantaneous LNP model neuron with
    receptive field kernel w.

    Parameters
    ----------

    w: np.array, (Dx * Dy, )
        (flattened) receptive field kernel.

    nT: int
```

```
        number of time steps

    dt: float
        duration of a frame in s

    R: float
        rate parameter

    v: float
        variance of the stimulus ensemble


    Returns
    -------

    c: np.array, (nT, )
        sampled spike counts in time bins

    r: np.array, (nT, )
        mean rate in time bins

    s: np.array, (Dx * Dy, nT)
        stimulus frames used

    Note
    ----

    See equations in task description above for a precise definition
    of the individual parameters.

    '''

    np.random.seed(42)

    # sampled spike counts array
    c = np.empty(0)
    # stimulus frames
    s = np.random.binomial(v,.5, size=(w.shape[0], nT))
    # mean rate
    r = np.exp(w.T@s)*dt*R


    # Poisson spike counts
    for i in r:
        c = np.append(c,np.random.poisson(lam=i))
    # ------------------------------------------------
    # Generate samples from an instantaneous LNP model
```

```
    # neuron with receptive field kernel w. (0.5 pts)
    # ------------------------------------------------

    return c, r, s

    ### variance of stimulus ensemble should be used, but cannot
    ### find any information about it ??
```

[74]:
```
D = 15      # number of pixels in one dimension,
            # the simulated RF here is a square
nT = 1000   # number of time bins
dt = 0.1    # frame rate, 0.1s per bin.
R = 50      # firing rate in Hz
v = 5       # stimulus variance

w = gen_gauss_rf(D,7,(1,1))
w = w.flatten()

c, r, s = sample_lnp(w, nT, dt, R, v)

print(r)
```

```
[ 45.16079112  61.2444825   57.92825726  70.57315883  51.03841263
  69.28122586  51.57503587  47.98142672  73.67549152  60.80166229
  36.19769933  61.61781181  50.51765486  75.82653885  68.42280142
  70.03707651  51.07818213  70.69517195  58.70229449  59.15217314
  51.99673135  58.65478548  69.03614416  67.70544604  62.0756533
  59.16600587  79.74341343  50.55251172  68.75874643  78.07916502
  75.18731366  63.34370384  74.03894968  51.30507619  51.987317
  51.00122665  46.40165604  71.58532174  56.29757445  64.10962769
  61.81287874  71.13143504  49.01688991  64.29248362  59.12257075
  61.78789427  56.73926546  66.57981907  61.95823976  88.04126598
  56.08286162  55.20293708  54.45155882  54.99377411  57.55592285
  50.17627773  40.35399913  45.76817337  72.81049238  55.16615834
  52.68164192  69.45792936  63.71786851  56.86658759  57.47433634
  68.96741661  48.79396381  50.511963    48.80983008  64.25739363
  70.6370607   73.87194704  61.14001364  64.4296986   81.58549585
  50.19563472  57.95015105  55.72805231  49.64874937  66.19899653
  63.23429517  74.04497319  60.41887163  71.32415429  76.23221357
  62.3816191   62.03771979  57.04705658  47.27700007  66.61813326
  61.97130376  46.86639261  87.87377696  64.64565502  58.56288509
  60.93311865  41.68000473  57.40290521  65.70154031  64.37529238
  54.41237409  46.88480874  61.97145198  67.17114875  54.68042318
  71.13028538  64.41631753  55.08248283  61.6020784   43.0292685
  61.41667511  44.79555967  68.1843978   63.32965658  56.30640152
  61.18748738  57.19761595  55.99533904  61.22806996  82.19799243
  63.29214525  68.86407314  56.60873813  65.06789888  56.1151765
  59.59620702  70.62038289  48.41531628  42.01391558  64.55741919
```

63.44068333	52.94629078	85.04811396	56.43719187	47.80957939
53.57696933	60.73879305	63.63328045	55.78699977	62.6114615
72.21731139	55.02458437	59.78700493	64.13734431	65.18191357
80.72693803	75.46960467	67.87996229	61.72450492	46.20104896
84.44376235	70.02643533	90.24175627	56.98363662	59.4475462
54.49748741	48.08843157	43.19932669	67.54024099	60.09086606
70.42574603	56.96035563	60.91972082	64.94342732	60.59504128
64.17726555	65.27623182	69.64077535	45.75734243	52.39273135
87.24721455	67.58006115	52.21895154	57.19899656	52.81366341
57.88887411	47.20931181	57.62296663	88.68999187	53.95437234
57.02896819	46.51293728	53.54950149	61.83006675	55.21641353
52.44336289	55.91593112	59.1731481	66.1797392	51.65304065
61.47648923	89.55668012	51.61182596	66.80981772	65.08955225
48.40560183	63.29322559	62.60339029	50.0480835	72.64072963
67.18093725	59.38664948	75.56100244	43.28087617	65.21052512
56.49050615	68.72396625	49.21271456	41.59298064	53.65313106
51.62854402	62.49459044	62.53051335	45.4924815	81.21781681
58.18248391	56.49637891	60.59655177	63.76349601	58.06381534
64.05813803	87.94483913	51.36180377	70.59653743	80.94491622
57.67627802	77.5438831	71.17760502	57.08886208	44.39369209
66.81404544	57.88260561	55.50565937	55.46852005	52.13866512
53.96392924	55.366951	60.57108917	59.09085084	50.08434667
66.49499127	62.41137063	43.86245048	50.67085965	70.85273266
56.48533823	61.06333803	71.55103296	55.76534576	46.50828945
64.30385427	60.27991351	50.01148097	53.13218926	69.13090465
49.17847693	71.13656423	44.50194801	61.42712556	70.54535185
71.65116606	50.52611887	64.2968411	66.65900769	61.47498791
79.53411405	58.98799806	66.40438648	55.87304746	71.32491358
78.66910637	46.8998912	76.94644537	54.25572067	67.77276731
47.24523248	77.34831427	71.04290796	74.54023676	57.6705175
49.83742586	67.40882004	65.60554461	64.12311353	78.09455818
45.72470545	64.69244303	88.83763519	43.32417787	64.68882987
45.14126134	65.12905497	51.55329874	65.55173191	68.14471936
49.24272389	53.3018446	46.79456947	53.24793779	49.75975058
59.41762742	76.95384831	60.61957679	52.99176578	66.69717299
67.07200587	68.68875707	48.60993459	66.85466342	44.51478697
80.28788101	52.94147397	46.47410335	65.36543178	59.88963294
55.60222324	61.82044853	61.01984802	49.64645783	75.2060576
66.14497314	71.93160382	60.41442541	59.99544027	61.0744101
65.43821446	43.05573313	51.11595549	52.35065282	66.26862008
70.33382436	50.40214957	59.59363613	89.65566338	56.11612939
83.67320584	54.06432858	69.11211369	63.24321003	67.19115455
56.94254268	69.61347526	49.81327668	50.81648789	50.1031974
72.5798569	69.01477789	59.98827234	68.49500106	67.28150518
47.17225061	58.60053882	88.38752272	70.44972106	47.90244672
63.5662571	72.79476822	64.97523639	61.08737638	56.11721557
76.84820946	79.1623403	71.4032966	55.49577546	87.16209447
85.61812563	63.76030146	56.01707513	57.28091898	57.36914049

73.46157562	45.52132768	47.61548843	94.25458987	64.42505094
71.13896394	49.08051241	48.64671551	61.64376955	41.21564672
66.5957048	93.94306766	68.23250632	57.40344865	58.51629975
59.97469143	52.04220714	66.47340786	57.61784621	47.10761736
56.85439577	49.08309718	56.86658435	51.44912922	53.24965223
69.09941354	92.57320679	68.42555663	56.9810587	72.02398855
65.56070115	62.27017663	68.61751521	92.11506781	50.5790157
53.57592469	64.38806566	55.36430503	48.85733861	72.61785731
61.99222838	64.63621792	42.11379058	67.44912288	73.54565901
66.95580343	81.46500289	75.84177335	50.90215285	86.6587992
74.45644637	55.3653309	80.496814	59.03036456	63.44659859
47.08387655	65.40832167	35.05547083	67.87211987	57.0537567
63.08628452	58.13652661	59.40389502	52.80749941	41.74554422
83.02424663	71.99710153	69.92861049	62.06242427	57.06457959
58.18056398	42.3071833	62.54208717	70.50355646	59.56176564
59.84162862	77.54268431	53.44109356	55.11639568	67.99970044
53.35921061	72.53565899	59.74023925	80.25363214	57.31035256
55.54760797	48.82529667	58.52122859	93.45591793	51.7181606
57.97119263	61.24910231	71.90542928	38.54112778	69.44591233
64.46621914	60.10695539	54.62266477	74.24516548	66.79375996
63.62716694	107.01270336	50.07687264	47.57003709	65.83138571
69.39439133	59.98108149	61.07164748	74.37033938	76.60597079
44.90020298	49.71603081	60.33617407	48.82265516	63.67036577
63.9993372	58.65434544	97.31534775	56.05988908	76.46800099
57.09418456	52.9531681	80.83260651	53.72411293	56.03667244
69.76674592	54.34270781	63.94208143	64.71713656	72.59403275
62.8843456	47.18316369	55.02337335	72.33888861	88.20331899
70.19235154	72.58878702	67.23227198	70.73417925	54.69717635
67.40320486	70.26140025	88.31325582	65.29053295	80.02535509
63.61652512	71.5122324	84.16181449	60.28541396	55.35850681
58.15308212	52.81912927	75.3061973	45.46556453	65.82340275
54.84338977	47.02377184	58.04655703	67.08408612	62.55324097
70.96985057	45.13814736	69.29707996	58.53235046	68.77373708
55.83728924	54.18086051	66.47689543	76.51871421	54.14223979
65.20190781	65.63451599	76.72253401	69.05323106	46.74287376
72.13550352	55.94790729	73.07644057	60.35667654	69.33767981
61.5081845	54.52275007	69.51263342	65.51906925	78.38600973
57.94789644	63.84106912	60.44828575	44.33517094	66.36569483
74.50075857	50.49860198	57.24283692	69.64636524	69.56184782
57.19817567	77.95933518	62.41738598	82.91457113	53.44566404
66.68137608	63.55434019	69.56371904	57.72298461	57.15258776
75.14746675	75.13723849	53.50555402	51.03832797	58.71000149
76.61801099	61.71704869	59.59001944	63.66974875	65.2259287
60.38381099	58.46610326	48.55074211	45.84719598	72.02761553
72.37728297	55.79071539	50.18150388	71.81235406	51.06686975
55.51970973	61.21766699	60.12669563	69.61902877	68.62657242
79.34112825	63.63720278	76.92476895	58.24573278	62.19949247
53.03456464	59.37629224	50.89083693	70.48485863	66.15003749

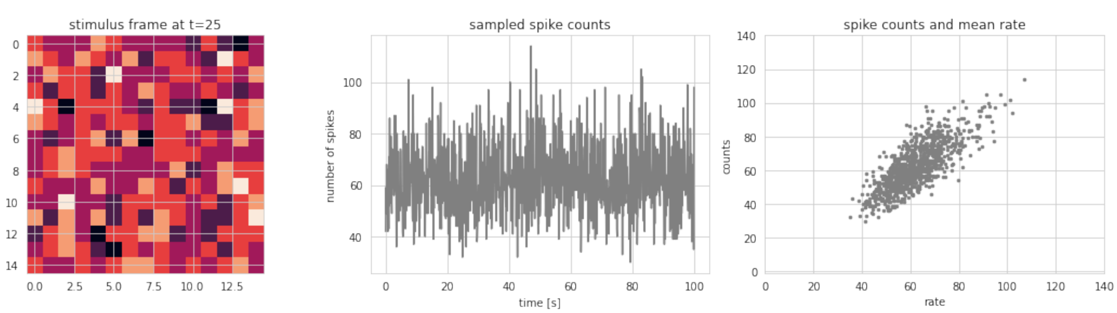
62.29321346	70.56771767	57.9490008	63.66081488	67.28882747
61.26430002	52.63053452	58.48360132	61.52063985	78.87195082
64.73277847	69.7950801	67.4423688	68.09891754	49.96963495
63.96289197	62.04976875	51.43687502	73.7472418	43.89567836
55.70059672	55.73471131	52.24324626	44.69185673	51.24296652
71.06946429	53.79315094	56.93788118	71.0665044	48.63692795
51.9510745	67.42700429	57.95831935	60.38594613	76.13266828
72.23351932	73.00084252	60.0736056	55.34615287	51.58497674
60.48728687	74.70158507	101.95556496	51.9816346	69.72973181
64.18692779	41.39068418	57.46063443	42.29221561	60.07254697
55.29664483	64.46459769	55.68613028	61.60677477	68.53988184
64.9003419	39.80768165	52.94607671	57.52919942	75.77545299
66.76381362	68.1030888	56.33555537	51.54452105	71.26100249
60.4253287	56.63201311	94.84580488	55.0210812	44.27067593
72.9554557	64.2461937	65.20741208	57.96359593	57.36205975
60.43939931	60.32697572	79.55657036	67.54749053	65.60224784
74.14775019	50.72182169	59.66684706	68.59663139	66.72155944
69.27616621	57.73069925	66.67778903	60.66016125	64.19340494
49.99616443	61.83253549	57.86103425	63.56456546	62.37941142
61.28069285	78.90640171	60.71577359	67.72271613	57.81847859
55.57386297	92.76357912	56.47118452	51.26498143	63.3391743
40.00658346	41.45058657	61.0441054	64.92113227	59.64491981
51.4874696	55.54233191	59.72212768	51.72489285	58.09483672
61.92968254	55.15878815	54.1152363	62.5592653	65.16787951
56.34856317	49.9134537	54.71437011	54.03551508	58.79687286
47.81001309	60.87478467	55.90519162	68.8646048	73.79228522
51.00192133	58.35052909	57.28762838	54.99368328	64.20284437
48.58255903	60.42504691	53.85641944	70.97105507	48.29969153
49.57233668	70.06654866	59.97955054	70.21391255	58.76968
53.42734703	89.05946302	65.48502064	58.81969316	62.16716178
68.80322147	60.27664692	69.29701675	80.12409156	66.35874626
51.88974186	67.71346045	39.31175147	60.10959492	40.48372774
51.74783521	60.59797418	58.26059077	51.21730102	77.75257525
66.72700256	78.57177482	58.08476129	64.26534185	50.95675125
73.7325217	59.52830638	68.36344356	66.13917747	53.77133414
63.99247294	56.06224074	60.47871228	68.19346063	45.43438035
51.25304792	53.81050326	92.33513717	41.54487421	54.94691245
45.0356263	65.15081958	75.64459433	50.51562589	63.83166675
67.77400492	71.84957128	69.27745123	72.72795937	50.72875451
50.27665166	57.64224009	64.78638533	59.45980985	74.73034579
62.11661716	51.79800986	47.75454606	81.43978538	70.03664403
51.64095141	54.48101755	54.84262952	54.30044641	63.16594612
48.79728137	67.48602585	79.90953427	71.51162124	66.68327182
63.33733449	51.78024085	56.55291638	91.3361689	55.22904094
52.55257224	73.59231396	101.15564206	58.5013188	66.11152084
56.32110311	68.71906711	69.19005547	58.20009031	54.92436481
84.00869481	66.13805951	91.40789349	41.9180008	54.59127595
70.21819338	57.3106766	81.60963214	50.51283241	58.19709671

```
 68.56750005  75.56303538  58.212451    64.31208714  54.31138224
 64.05535609  52.26029035  71.59603232  69.17539581  72.86044417
 54.65943651  48.97866177  57.68616477  71.20628544  53.8789052
 68.34476269  53.79225003  53.59539158  68.8648707   44.6305188
 64.815917    73.91444659  45.63790331  67.89176678  55.99973485
 53.16742474  61.00139252  73.79885842  66.22557778  62.86530302
 59.01217598  59.19203102  70.75924662  64.08486772  68.55306867
 59.99220776  66.89030941  56.76794327  55.83945812  61.93653624
 81.75534174  62.41793717  62.41668151  60.09121827  60.41847817
 73.95779295  58.42621268  72.34408844  58.87322412  65.73709275
 76.99642083  54.66428103  57.76404736  50.50968562  66.24322909
 52.55373021  50.23435799  59.67317554  45.99261258  60.41102436
 66.78356511  58.98362028  66.90903648  53.69267737  72.22766033
 79.50983294  50.28727096  59.99375101  63.97934559  54.49285391
 76.86600508  52.89559321  71.06311976  70.69530915  47.16104073
 57.70619923  54.00515216  70.13379886  69.95551058  61.77482636
 56.68679634  53.55279301  59.9103346   72.1763101   42.02253602
 66.9682372   66.26138158  52.59582735  53.99500645  66.24954331
 82.11329909  60.30266523  55.85380784  44.11848611  61.69678255
 75.24103348  53.92751841  62.84507565  49.17936256  52.9221562
 67.96199304  80.5335813   65.5903656   71.45035478  49.85273191
 60.34320571  53.28354496  45.38882376  66.14970379  54.79864225
 62.7927989   56.17333188  60.60721678  61.60652394  68.57911984
 75.16812931  65.32476633  63.42748546  55.08563758  75.48795093
 68.41728658  57.87423659  61.76619416  61.61211353  70.11977009
 57.73117603  55.19491717  70.60646526  98.70918857  54.92074294
 53.64424853  69.63803834  55.48078047  64.62749403  63.17567559
 56.87259707  66.73762793  60.26666417  83.61176555  62.80350723
 67.00782073  67.98014269  52.29927303  62.70882125  57.22994511
 62.04673159  69.53487734  62.33837778  44.15173983  66.41555065]
```

Plot the responses of the cell.

[75]:
```python
# insert your code here

# --------------
# Plot (0.5 pts)
# --------------
t = np.arange(0,nT) * dt
#plt.figure(figsize=(15,4))

fig, axs = plt.subplots(1, 3,figsize=(15,4))
# -----------------------------------------------
# (1) one example frame from `s`;
# -----------------------------------------------

axs[0].imshow(s[:,25].reshape((D,D)))
axs[0].set_title('stimulus frame at t=25')
```

```
# ------------------------------------------------
# (2) the simulated response `c`;
# ------------------------------------------------
axs[1].plot(t, c, color='grey')

axs[1].set_ylabel('number of spikes')
axs[1].set_xlabel('time [s]')
axs[2].set_ylim(0,140)
axs[1].set_xlim(-5,105)
axs[1].set_title('sampled spike counts')



# ------------------------------------------------
# (3) a scatter plot of `r` and `c`;
# ------------------------------------------------

axs[2].scatter(r,c,s=6,color = 'grey')
axs[2].set_ylabel('counts')
axs[2].set_xlabel('rate')
axs[2].set_xlim(0,140)
axs[2].set_ylim(-1,140)
axs[2].set_title('spike counts and mean rate')




plt.tight_layout()
plt.show()
```



1.1.3 Implementation

Before you run your optimizer, make sure the gradient is correct. The helper function `check_grad` in `scipy.optimize` can help you do that. This package also has suitable functions for optimization. If you generate a large number of samples, the fitted receptive field will look more similar to the true receptive field. With more samples, the optimization takes longer, however.

[76]:
```python
def negloglike_lnp(x, c, s, dt=0.1, R=50):
    '''Implements the negative (!) log-likelihood of the LNP model and its
    gradient with respect to the receptive field w.

    Parameters
    ----------

    x: np.array, (Dx * Dy, )
      current receptive field

    c: np.array, (nT, )
      spike counts

    s: np.array, (Dx * Dy, nT)
      stimulus matrix


    Returns
    -------

    f: float
      function value of the negative log likelihood at x

    df: np.array, (Dx * Dy, )
      gradient of the negative log likelihood with respect to x
    '''

    p_1 = np.sum(c*np.log(dt*R))
    p_2 = np.sum(np.log(sp.special.factorial(c)))
    # use neg loglikelihood
    f = -c@(x.T@s)
    f += -(p_1+p_2)
    f += np.sum(np.exp(x.T@s)*dt*R)
  # Compute gradient
    d_1 = c@s.T
    d_2 = dt*R*np.exp(x.T@s)
    df = -(d_1 - d_2@s.T)
    # ------------------------------------------------
    # Implement the negative log-likelihood of the LNP
    # and its gradient with respect to the receptive
    # field `w` using the simplified equantions you
    # calculated earlier. (0.5 pts)
    # ------------------------------------------------
```

```
    return f, df
```

Fit receptive field maximizing the log likelihood

[77]:
```python
f, df = negloglike_lnp(w, c, s)

# function derivative to be checked
f = lambda w: negloglike_lnp(w, c, s)[0]
# gradient to be checked
df = lambda w: negloglike_lnp(w, c, s)[1]
# check if gradient is correct
print(opt.check_grad(f,df,w))

# Estimate receptife field rf
random_rf = np.random.randint(2,size=(D*D))/1000
estimated_rf = opt.minimize(negloglike_lnp,random_rf,args=(c,s),jac=True).x
# ------------------------------------------
# Estimate the receptive field by maximizing
# the log-likelihood (or more commonly,
# minimizing the negative log-likelihood).
#
# Tips: use scipy.optimize.minimize(). (1 pt)
# ------------------------------------------
```

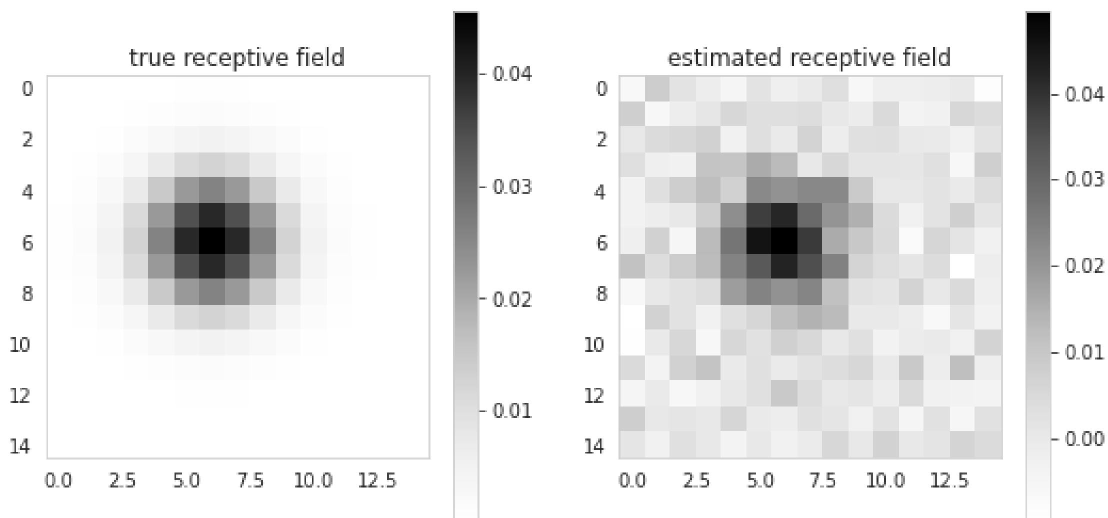
```
0.0266871430454665
```

[78]:
```python
fig, axs = plt.subplots(1,2,figsize=(10,5))


p1=axs[0].imshow(w.reshape((D,D)), cmap='Greys')
axs[0].grid(False)
axs[0].set_title('true receptive field')

p2=axs[1].imshow(estimated_rf.reshape((D,D)), cmap='Greys')
axs[1].grid(False)
axs[1].set_title('estimated receptive field')


plt.tight_layout
plt.colorbar(p1,ax=axs[0])
plt.colorbar(p2,ax=axs[1])

# ------------------------------------
# Plot the ground truth and estimated
# `w` side by side. (0.5 pts)
# ------------------------------------
```

```
/home/v/.local/lib/python3.7/site-packages/ipykernel_launcher.py:14:
MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh()
```

```
is deprecated since 3.5 and will be removed two minor releases later; please
call grid(False) first.

/home/v/.local/lib/python3.7/site-packages/ipykernel_launcher.py:15:
MatplotlibDeprecationWarning: Auto-removal of grids by pcolor() and pcolormesh()
is deprecated since 3.5 and will be removed two minor releases later; please
call grid(False) first.
  from ipykernel import kernelapp as app
```

[78]:
```
<matplotlib.colorbar.Colorbar at 0x7fb2b3eb36d8>
```



1.2 Task 2: Apply to real neuron

Download the dataset for this task from Ilias (`nda_ex_6_data.mat`). It contains a stimulus matrix (`s`) in the same format you used before and the spike times. In addition, there is an array called `trigger` which contains the times at which the stimulus frames were swapped.

- Generate an array of spike counts at the same temporal resolution as the stimulus frames
- Fit the receptive field with time lags of 0 to 4 frames. Fit them one lag at a time (the ML fit is very sensitive to the number of parameters estimated and will not produce good results if you fit the full space-time receptive field for more than two time lags at once).
- Plot the resulting filters

Grading: 2 pts

[79]:
```
var = io.loadmat('./data/nda_ex_6_data.mat')

# t contains the spike times of the neuron
t = var['DN_spiketimes'].flatten()
```

```
# trigger contains the times at which the stimulus flipped
trigger = var['DN_triggertimes'].flatten()

# contains the stimulus movie with black and white pixels
s = var['DN_stim']
s = s.reshape((300,1500)) # the shape of each frame is (20, 15)
s = s[:,1:len(trigger)]
```

Create vector of spike counts

[80]:
```
c,b = np.histogram(t,bins=trigger)
# ------------------------------------------
# Bin the spike counts at the same temporal
# resolution as the stimulus (0.5 pts)
# ------------------------------------------
```

Fit receptive field for each frame separately

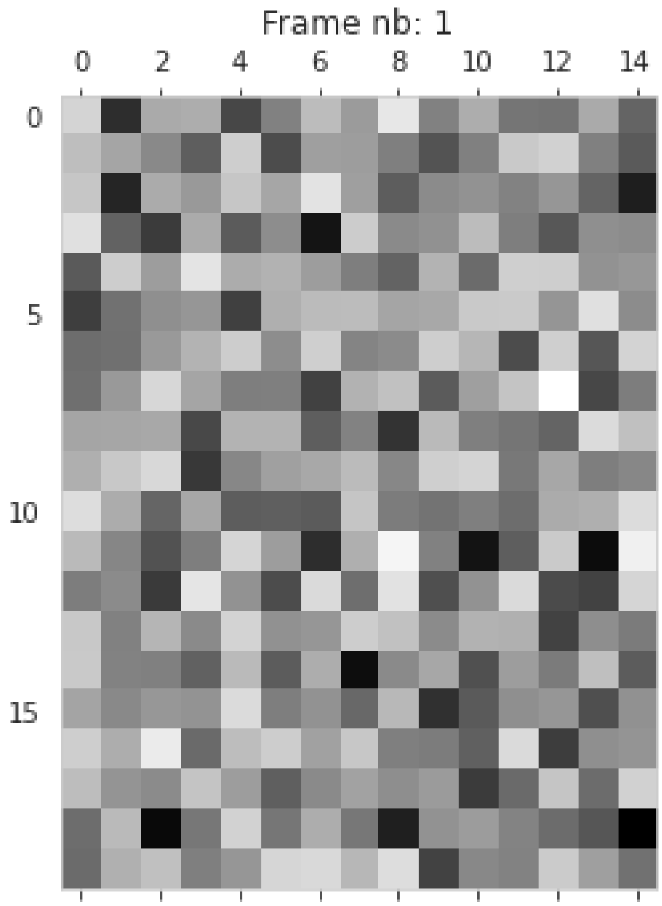
[81]:
```
nT = 1000  # number of time bins
lags = np.arange(5) # time lags of 0-5 frames
res = s.shape[0] # same as frame shape
D1 = 20 # number of pixels in one dimension
D2 = 15 # number of pixels in one dimension


random_rf = (np.random.uniform(0,1,size=res)/nT).flatten()
estim_rf_lags = np.zeros((len(lags),res))
# Use one lag at a time
for l in lags:
  estim_rf_lags[l,:] = opt.minimize(negloglike_lnp,random_rf,args=(c[l:],s[:,:
 ↪len(c)-l]),jac=True).x


# ------------------------------------------
# Fit the receptive field with time lags of
# 0 to 4 frames separately (1 pt)
#
# The final receptive field (`w_hat`) should
# be in the shape of (Dx * Dy, 5)
# ------------------------------------------
```

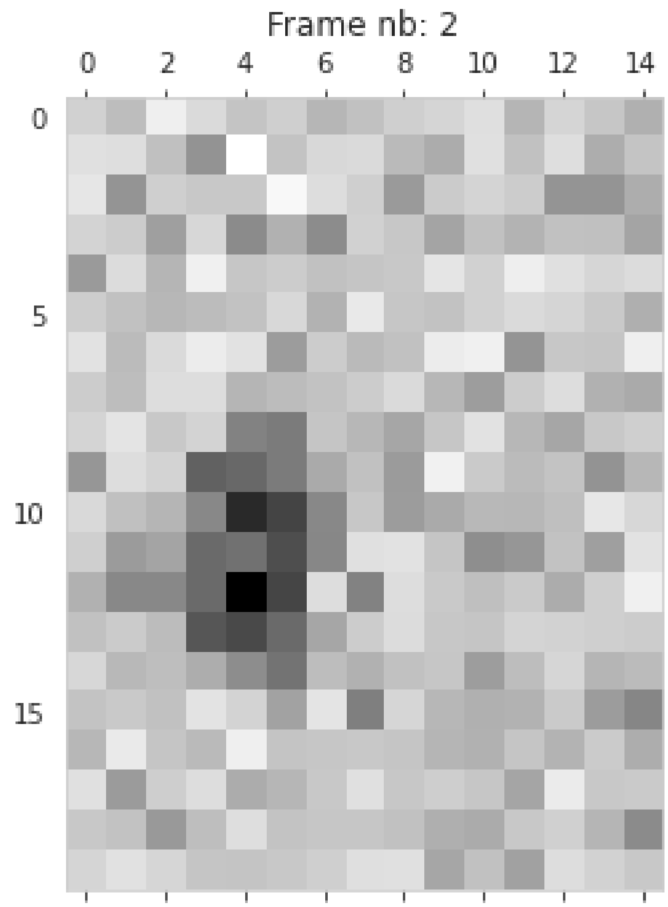
Plot the frames one by one

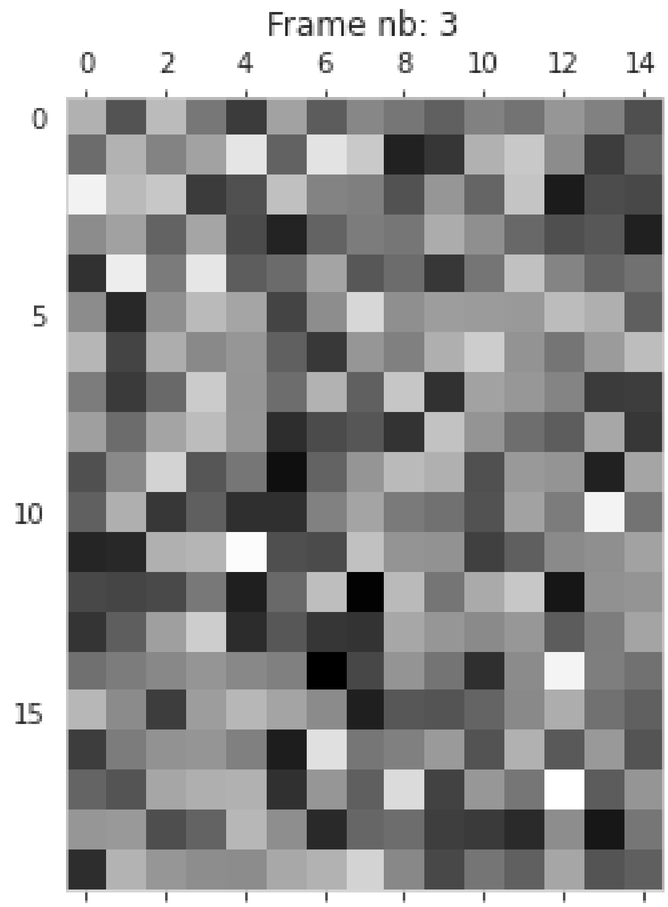
[82]:
```
matfig = plt.figure(figsize=(10,4))


for i in lags:
  plt.matshow(estim_rf_lags[i,:].reshape((D1,D2)), cmap='Greys')
```

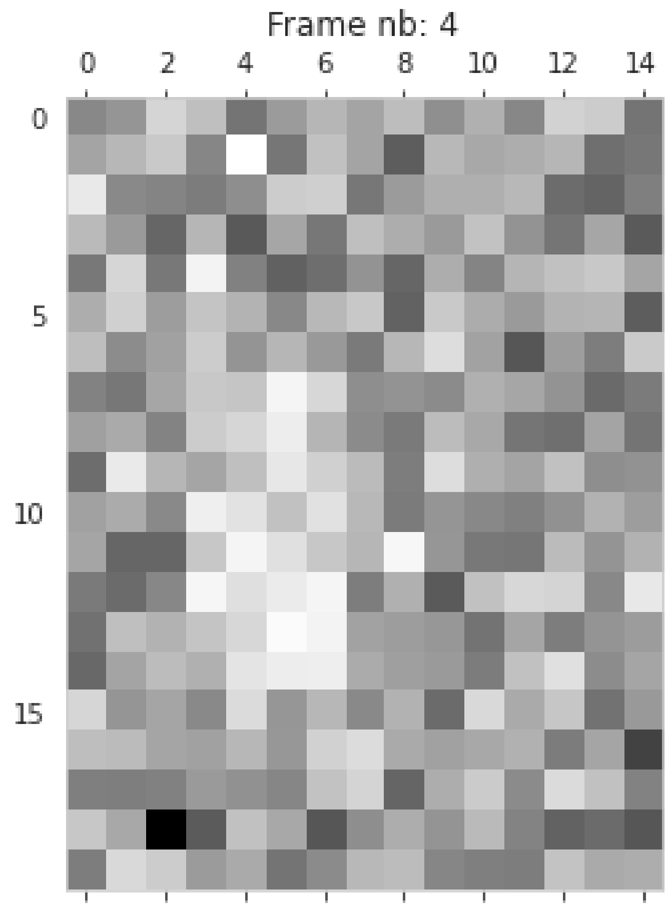
```
  plt.grid(False)
  plt.title("Frame nb: " + str(i+1))
  #plt.tight_layout()
# --------------------------------------------
# Plot all 5 frames of the fitted RFs (0.5 pt)
# --------------------------------------------
```

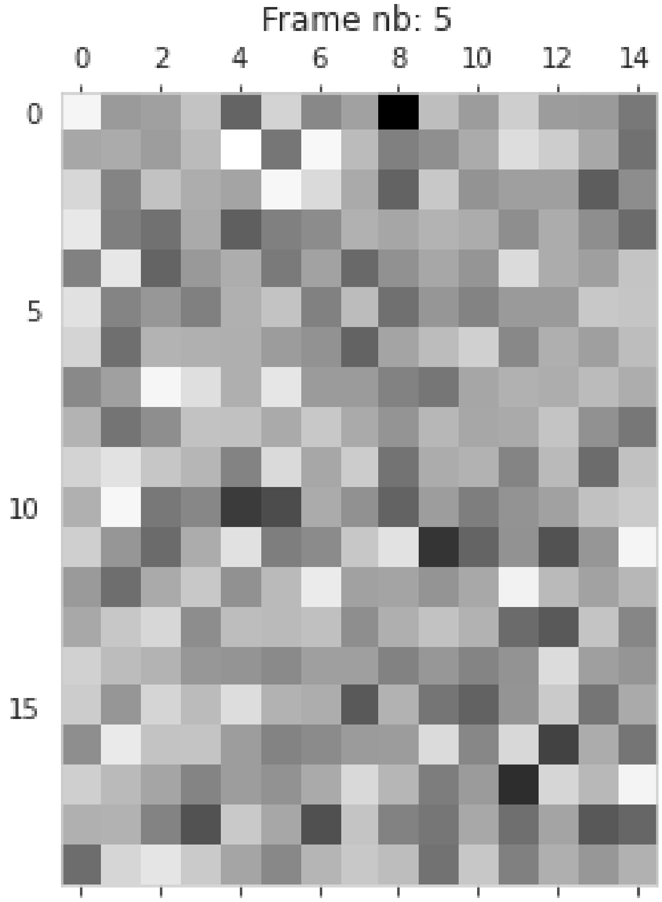
```
<Figure size 720x288 with 0 Axes>
```




Frame nb: 2
0
2
4
6
8
10
12
14
0
5
10
15


Frame nb: 3
0
2
4
6
8
10
12
14
0
5
10
15


Frame nb: 4
0
2
4
6
8
10
12
14
0
5
10
15



1.3 Task 3: Separate space/time components

The receptive field of the neuron can be decomposed into a spatial and a temporal component. Because of the way we computed them, both are independent and the resulting spatio-temporal component is thus called separable. As discussed in the lecture, you can use singular-value decomposition to separate these two:

$$W = u_1 s_1 v_1^T$$

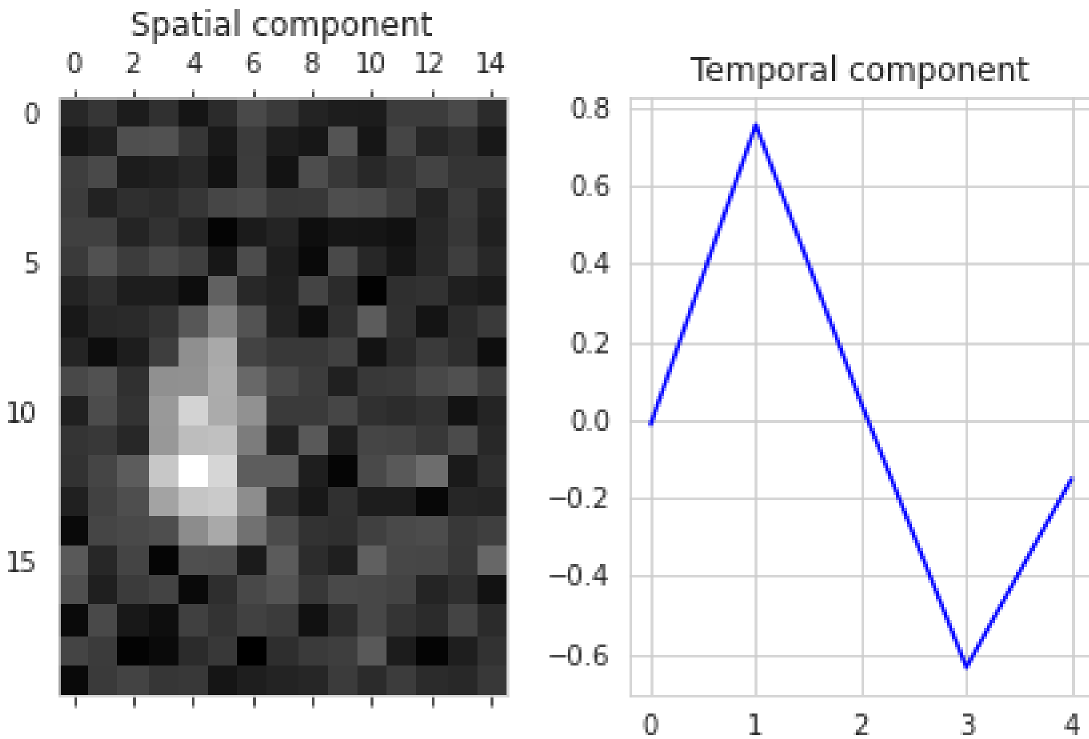
Here u_1 and v_1 are the singular vectors belonging to the 1st singular value s_1 and provide a long rank approximation of W, the array with all receptive fields. It is important that the mean is subtracted before computing the SVD.

Plot the first temporal component and the first spatial component. You can use a Python implementation of SVD. The results can look a bit puzzling, because the sign of the components is arbitrary.

Grading: 1 pts

[83]:

```
W = estim_rf_lags
# substract mean
W = W - np.mean(estim_rf_lags, axis=0)

# compute svd
u,_,v = np.linalg.svd(W)

# temporal component
u_temp = u[:,0]
# spatial component
v1 = v[0,:]
v_spatial = v1.reshape(D1,D2)

fig, axs = plt.subplots(1,2, figsize=(6,4), squeeze=True)

axs[0].matshow(v_spatial, cmap='gray')
axs[0].set_title("Spatial component")
axs[0].grid(False)

axs[1].plot(u_temp, color='blue')
axs[1].set_title("Temporal component")



plt.tight_layout()
plt.show()

# --------------------------------------------
# Apply SVD to the fitted receptive field,
# you can use either numpy or sklearn (0.5 pt)
# --------------------------------------------

# --------------------------------------------
# Plot the spatial and temporal components (0.5 pt)
# --------------------------------------------
```



1.4 Task 4: Regularized receptive field

As you can see, maximum likelihood estimation of linear receptive fields can be quite noisy, if little data is available.

To improve on this, one can regularize the receptive field vector and a term to the cost function

$$C(w) = L(w) + \alpha||w||_p^2$$

Here, the p indicates which norm of w is used: for $p = 2$, this is shrinks all coefficient equally to zero; for $p = 1$, it favors sparse solutions, a penality also known as lasso. Because the 1-norm is not smooth at zero, it is not as straightforward to implement "by hand".

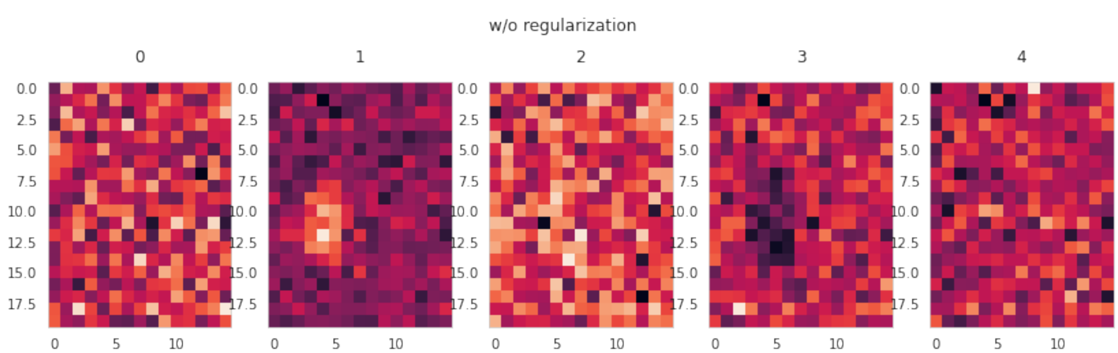
Use a toolbox with an implementation of the lasso-penalization and fit the receptive field. Possibly, you will have to try different values of the regularization parameter α. Plot your estimates from above and the lasso-estimates. How do they differ? What happens when you increase or decrease $alpha$?

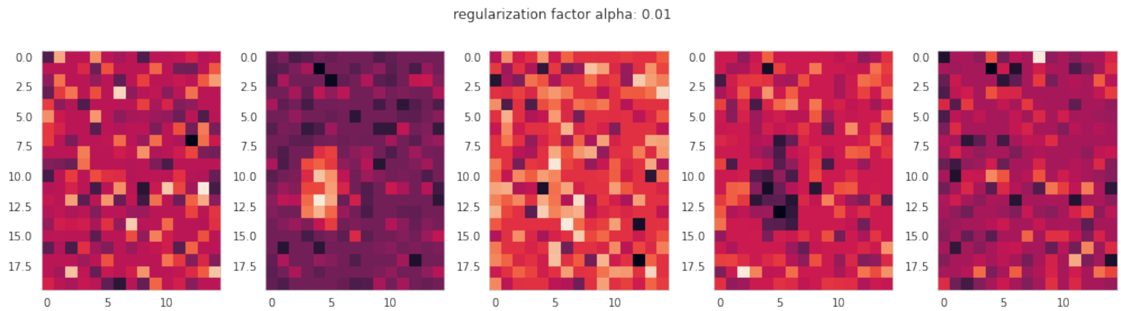
If you want to keep the Poisson noise model, you can use the implementation in `pyglmnet`. Otherwise, you can also resort to the linear model from `sklearn` which assumes Gaussian noise (which in my hands was much faster).

Grading: 2 pts

[87]:

```python
from sklearn import linear_model

alpha = 0.01
fig,axs = plt.subplots(1,len(lags),figsize=(14,4))
fig.suptitle('w/o regularization')

for i,l in enumerate(lags):
  colors = axs[i].imshow(estim_rf_lags[l,:].reshape((D1,D2)))
  axs[i].set_title(str(l), pad = 15)
  axs[i].grid(False)


fig,ax = plt.subplots(1,len(lags),figsize=(14,4))
fig.suptitle('regularization factor alpha: '+str(alpha))

for i,l in enumerate(lags):
  model = linear_model.Lasso(alpha=alpha, selection="random")
  model.fit(s.T,c)

  lasso_rf_lags = np.zeros((len(lags),res))
  model.fit(s[:,:len(c)-l].T,c[l:])

  lasso_rf_lags[l,:] = model.coef_
  ax[i].grid(False)

  colors = ax[i].imshow(lasso_rf_lags[l,:].reshape((D1,D2)))
  plt.tight_layout()



# ------------------------------------------
# Fit the receptive field with time lags of
# 0 to 4 frames separately (the same as before)
# with sklern or pyglmnet (1 pt)
# ------------------------------------------
```





For alpha=0.01 the lasso estimation fits the estimated filter quite good. This does not noticably change with slightly decreasing values for alpha. With a larger value for alpha the estimation gets less accurate. If alpha is chosen too small, the result gets worst.

[85]:
```
# insert your code here


# Included in the winodw above


# --------------------------------------------------
# Plot all 5 frames of the fitted RFs, compare them
# with the ones without regularization (0.5 pt)
# --------------------------------------------------
```