training-and-testing

December 10, 2023

[35]:
```
import numpy as np
import json
import os

from tensorflow.keras.preprocessing import sequence
```

[36]:
```
sequences_file = os.path.join('..', 'data', 'protein-seqs-2021-05-11-122938.
 ↪txt')
functions_file = os.path.join('..', 'data',␣
 ↪'protein-functions-2021-05-11-122938.txt')
```

[37]:
```
with open(functions_file) as fn_file:
    has_function = json.load(fn_file)
```

[38]:
```
has_function  # just to see what we have loaded
```

[38]:
```
['P27361',
 'P53779',
 'Q9UHC1',
 'Q9NYL2',
 'O15440',
 'P33527',
 'Q92887',
 'O15438',
 'O15439',
 'Q5T3U5',
 'P42345',
 'O75648',
 'Q16659',
 'Q8NB16',
 'Q02750',
 'O95255',
 'O95396',
 'O43196',
 'P46734',
 'P49914',
 'Q6DT37',
```

'Q9H3H1',
'Q9HCE1',
'P52564',
'Q9Y5S2',
'Q96J65',
'P20585',
'O15457',
'P52701',
'Q5VT25',
'A7E2Y1',
'Q9UKN7',
'Q9Y6X6',
'Q96MN2',
'P13535',
'Q32MK0',
'O43795',
'O00159',
'Q9UM54',
'Q13402',
'Q9UKX2',
'P13533',
'Q9H1R3',
'Q8WXR4',
'Q6PIF6',
'B2RTY4',
'Q6IA69',
'Q86W25',
'Q9UKX3',
'Q7Z406',
'P12882',
'Q15746',
'O15146',
'Q9Y623',
'P12883',
'P35579',
'B0I1T2',
'Q9Y4I1',
'Q96JP2',
'Q8IUG5',
'Q9UBC5',
'O00160',
'P53602',
'Q92614',
'P35749',
'P11055',
'Q9Y2K3',
'Q86YV6',

'Q9HD67',
'Q8N1T3',
'Q9ULV0',
'Q13459',
'Q86W26',
'P59046',
'Q9UJ70',
'Q86W24',
'Q13232',
'Q9H0A0',
'P22392',
'Q8IY84',
'Q86WI3',
'Q96P20',
'Q86UW6',
'Q8IVL1',
'O75414',
'Q8NG66',
'Q96PY6',
'P35580',
'Q96H55',
'O94832',
'Q12965',
'Q8NEV4',
'Q9NQX4',
'O95544',
'O60361',
'O00746',
'Q9NX02',
'Q8WX94',
'Q86W28',
'Q8IVL0',
'P15531',
'Q86SG6',
'Q9HC98',
'Q8TD19',
'Q4G0N4',
'P59045',
'P59047',
'Q7RTR0',
'Q9Y5B8',
'Q6P3R8',
'Q8TDX7',
'Q6ZWH5',
'P51955',
'Q86UT6',
'P51956',

'Q9UBE8',
'Q7RTR2',
'Q9NPP4',
'Q9BZQ4',
'P59044',
'Q8IW45',
'Q9HC29',
'Q9NPI5',
'Q9HAN9',
'Q5SY16',
'Q96T66',
'Q9Y239',
'Q9P0J0',
'P51957',
'Q14978',
'Q9C000',
'Q9UHY1',
'P46459',
'Q9BSD7',
'Q8TB37',
'Q16620',
'Q149M9',
'P29728',
'Q9H093',
'P04629',
'O60285',
'P53384',
'Q9Y5Y2',
'Q7Z2Y5',
'Q9NSY0',
'Q9NWW6',
'Q16288',
'O43929',
'O15381',
'Q15646',
'Q9NTK5',
'P00973',
'Q9Y6K5',
'Q5VST9',
'O43913',
'Q13415',
'Q9UBL9',
'P47900',
'P51582',
'O00443',
'O95747',
'O00750',

'P04637',
'P51575',
'Q9NVV4',
'Q96RG2',
'O14841',
'P54886',
'Q58A45',
'Q13177',
'O96013',
'Q99571',
'O15547',
'O75747',
'Q8TCG2',
'O75914',
'Q9P286',
'Q8TE04',
'Q9NVE7',
'P51003',
'Q9BWT3',
'O95340',
'Q13153',
'Q9NQU5',
'Q9H999',
'Q9NRJ5',
'A6NDB9',
'P09619',
'P00558',
'P07205',
'P05166',
'P16234',
'P05165',
'Q15645',
'Q15119',
'O15530',
'Q9H792',
'Q96RR1',
'P56373',
'Q93086',
'Q99572',
'Q9BTU6',
'Q9BZ23',
'O00764',
'P30086',
'Q15120',
'Q6A1A2',
'Q9NTI5',
'Q8IZE3',

'O43252',
'O60331',
'Q9NP80',
'P42336',
'Q504Y2',
'P53350',
'P48426',
'Q9UBF8',
'O14986',
'P11309',
'Q16512',
'Q9H4B4',
'O00444',
'Q96T60',
'P54277',
'P54278',
'Q15126',
'Q99640',
'Q6P5Z2',
'Q9BXM7',
'Q8NEB9',
'Q496M5',
'O43930',
'Q96S44',
'P11908',
'P17980',
'P43686',
'P53041',
'P51817',
'P21108',
'P62191',
'Q92620',
'P11498',
'Q92878',
'O75771',
'O75943',
'Q92698',
'Q9Y620',
'O43502',
'P04049',
'Q9H477',
'P22102',
'Q8IV42',
'Q13882',
'P35998',
'P62195',
'P78527',

'Q13523',
'P60891',
'P62333',
'P27708',
'Q13308',
'Q15257',
'O15067',
'P06737',
'P22234',
'Q9NRF8',
'P17812',
'O15315',
'Q06609',
'Q9BVS4',
'O43353',
'P46063',
'P51606',
'P57078',
'O94761',
'O94762',
'Q15835',
'P35250',
'P07949',
'P40937',
'O94955',
'Q9ULI2',
'Q92900',
'P35251',
'P35249',
'Q8IXN7',
'Q13546',
'Q969G6',
'Q9BRS2',
'O14730',
'Q9Y572',
'P23921',
'Q05823',
'Q13464',
'Q04912',
'Q9Y6S9',
'O75116',
'Q52WX2',
'Q9NZ71',
'Q13761',
'Q9Y230',
'P34925',
'O76082',

'O14975',
'Q9Y2P5',
'Q9UBT2',
'P0C264',
'Q9H015',
'Q13950',
'Q9Y265',
'Q9Y3I0',
'Q96KG9',
'P0C263',
'O00442',
'Q01196',
'O00141',
'Q9UQD0',
'Q6P3W7',
'P21675',
'P17987',
'Q9H2K8',
'Q03519',
'P78371',
'P42680',
'A2RTX5',
'P23381',
'P54577',
'Q96SF2',
'Q15569',
'P26639',
'Q9BW92',
'Q9Y2Z4',
'Q8TEA7',
'O43776',
'P26640',
'Q9UL54',
'Q7L3T8',
'P47897',
'P13984',
'P50990',
'Q9UHD2',
'P50991',
'P48643',
'A6NM43',
'Q92526',
'P40227',
'Q99973',
'Q03518',
'Q587J7',
'P55072',

'Q9H497',
'Q02763',
'Q12931',
'O43615',
'Q9UKE5',
'Q9ULW0',
'Q86UE8',
'O14657',
'Q02880',
'Q9H3S4',
'O75962',
'Q96KB5',
'O14656',
'Q9Y2W1',
'Q5JU69',
'P11388',
'Q96Q11',
'Q96QT4',
'Q96PN8',
'Q6IQ55',
'O95922',
'P42681',
'A1L167',
'Q9NXH8',
'Q96PF2',
'Q8TD43',
'Q9BX84',
'Q8NER1',
'Q9BXA6',
'Q96RU7',
'Q9HBA0',
'Q3LXA3',
'Q13470',
'Q8N2E6',
'Q9BXA7',
'Q5TCY1',
'Q9Y4R7',
'A6PVC2',
'Q8NG68',
'P62253',
'Q9Y385',
'P49459',
'O00762',
'Q9H832',
'Q6SA08',
'Q9UNY4',
'P33981',

'Q6IBS0',
'O75643',
'Q9Y2X8',
'P51965',
'Q8N2K1',
'Q7Z7E8',
'Q712K3',
'Q9GZZ9',
'Q9NPD8',
'Q6ZT98',
'Q3SXZ7',
'Q96LR5',
'Q8WVN8',
'P49427',
'Q8TBC4',
'P63146',
'Q9C0C9',
'Q96B02',
'P62837',
'P61077',
'Q969M7',
'O95155',
'A0AVT1',
'Q9HA47',
'P29597',
'P51668',
'Q969T4',
'P68036',
'P61081',
'P63279',
'Q9NWZ5',
'Q8NHH1',
'Q9BWV7',
'Q14679',
'P22314',
'P62256',
'P61086',
'P61088',
'Q16763',
'Q5VVX9',
'Q9BZX2',
'P30530',
'O75385',
'Q96C45',
'A6NNM8',
'Q6EMB2',
'Q8N841',

'Q12792',
'Q06418',
'P60604',
'P41226',
'P21281',
'O43314',
'Q8IYT8',
'Q6ZVT0',
'Q14166',
'Q8TAS1',
'Q6PHR2',
'Q6PFW1',
'P15313',
'P35916',
'Q96J92',
'Q96S55',
'Q15904',
'A3KMH1',
'P17948',
'P35968',
'Q9Y4B6',
'P30291',
'P0C1S8',
'P38606',
'Q96TA2',
'Q5FWF4',
'Q9UGI9',
'P54646',
'Q96GR2',
'P42684',
'Q08AH3',
'P54619',
'Q4W5N1',
'Q86UK0',
'Q9NUT2',
'Q9NRK6',
'Q9NSE7',
'P33897',
'P28288',
'Q9H222',
'O95477',
'Q8WWZ7',
'Q8IUA7',
'Q86UQ4',
'O75027',
'Q9NP78',
'O95342',

'Q13085',
'P53396',
'P33121',
'Q9ULC5',
'Q13131',
'Q8N139',
'Q9H221',
'Q09428',
'Q96J66',
'Q9UBJ2',
'Q9UNQ0',
'Q8IZY2',
'Q2M3G0',
'Q9NP58',
'Q9BZC7',
'Q99758',
'P78363',
'O94911',
'Q8WWZ4',
'O60706',
'O14678',
'Q9UG63',
'P00519',
'Q2M2I8',
'P61221',
'Q8NE71',
'Q9NUQ8',
'P45844',
'Q9H172',
'Q86TW2',
'P51828',
'Q53H12',
'P31749',
'Q4L235',
'P68032',
'Q08462',
'P60709',
'P63261',
'P37023',
'Q8NFM4',
'O43306',
'Q4G176',
'Q04771',
'Q68CK6',
'Q9NR19',
'Q96CM8',
'Q562R1',

'P61163',
'Q07912',
'Q7Z695',
'Q9Y4W6',
'Q5FVE4',
'P42025',
'Q8WYK0',
'Q6P461',
'P62736',
'P68133',
'O60266',
'P40145',
'Q86V21',
'Q9UGJ0',
'Q9BTE6',
'O00763',
'Q08AH1',
'Q53FZ2',
'P0C7M7',
'Q8NER5',
'O95573',
'O60488',
'Q9UKU0',
'Q6NUN0',
'Q9H6R3',
'Q9BYX7',
'P63267',
'P36896',
'Q96PN6',
'P31751',
'Q86TB3',
'Q96QP1',
'Q9Y243',
'Q08828',
'Q9NUB1',
'O95622',
'Q9UM73',
'O60503',
'Q16671',
'P16066',
'P20594',
'O14727',
'P10398',
'P25098',
'P55263',
'Q96L96',
'Q8NFD2',

'Q9Y4B4',
'P13637',
'Q16720',
'Q5T9A4',
'Q8NB49',
'P05023',
'Q13733',
'Q9ULI0',
'P00966',
'Q9P1U1',
'P61158',
'P35626',
'P61160',
'Q9Y2Q0',
'O43520',
'O60423',
'Q96QE3',
'O43681',
'O60312',
'P98196',
'P54707',
'Q4VNC0',
'P50993',
'O14983',
'P06576',
'Q9C0K3',
'Q9P241',
'Q9Y2G3',
'Q9NQ11',
'Q13315',
'P20020',
'Q01814',
'P23634',
'P98194',
'Q9NVI7',
'Q9UQB9',
'Q8TF62',
'Q04656',
'P35670',
'Q9H981',
'Q9HD20',
'Q5T2N8',
'P20648',
'O75110',
'Q8N3C0',
'Q8NBU5',
'Q6PL18',

'P08243',
'O94823',
'Q9H7F0',
'Q4VNC1',
'P16615',
'Q93084',
'O75185',
'Q9NTI2',
'P25705',
'Q9UIG0',
'Q96GD4',
'Q6ZW61',
'P27037',
'P98198',
'O43861',
'O14965',
'O14874',
'P46100',
'Q13873',
'Q13705',
'Q8TAM1',
'P51451',
'Q14692',
'Q8IWQ3',
'O60566',
'Q13075',
'O00238',
'Q9Y276',
'Q13535',
'P11274',
'P54132',
'P15056',
'P11586',
'Q8NCB2',
'P51813',
'Q6UB35',
'Q9BRT8',
'Q4V339',
'Q9NSY1',
'P36894',
'P50747',
'Q00975',
'O43683',
'P33076',
'Q06187',
'Q8TDC3',
'O14981',

'Q00526',
'Q8IUF1',
'A6NM15',
'Q96Q40',
'Q9UQ88',
'Q8IVW4',
'Q5MAI5',
'Q8N884',
'Q9BWU1',
'Q00534',
'Q00532',
'Q8TCT0',
'P21127',
'P49336',
'P50750',
'Q92772',
'O76039',
'Q5RIA9',
'Q99741',
'O00311',
'O94921',
'Q07002',
'P24941',
'O14578',
'P19784',
'P49916',
'P49917',
'Q8IVF4',
'Q0VDD8',
'Q8WW22',
'Q96EY1',
'Q8TE96',
'O60884',
'P18858',
'O60231',
'Q7L2E3',
'Q7L7V1',
'Q14147',
'Q6P158',
'P51530',
'Q7L8W6',
'Q5JTY5',
'Q15131',
'Q9NYV4',
'Q14004',
'Q00536',
'P50613',

'Q02224',
'Q12873',
'Q8TD26',
'Q9UHD1',
'Q00535',
'O14757',
'Q00537',
'P06493',
'P11802',
'P13569',
'P61604',
'P10809',
'O96017',
'P51790',
'Q8IZL9',
'Q92989',
'Q5EBM0',
'Q14839',
'Q9HCK8',
'P51793',
'P51795',
'O14646',
'Q8TDI0',
'P35790',
'Q9Y259',
'P51797',
'P49760',
'Q86WJ1',
'O14647',
'Q9P2D1',
'Q3L8U1',
'Q9H078',
'P51798',
'P49759',
'Q9HAZ1',
'Q9H8M5',
'O76031',
'Q8NEV1',
'P68400',
'O14936',
'Q96D53',
'Q8NI60',
'P31327',
'P41240',
'Q13057',
'A5YM72',
'P49761',

```
'O43293',
'P07333',
'Q8IWT3',
'Q8WVB6',
'A8MPP1',
'Q9Y6G9',
'Q8WVC6',
'P53355',
'O43237',
'P26196',
'O15075',
'Q9NUU7',
'Q9NR30',
'Q7L014',
'Q9BQ39',
'Q86TM3',
'P17844',
'P23743',
'Q16760',
'P49619',
'P52824',
'Q8IX18',
'Q14562',
'Q13206',
'Q92841',
'Q92499',
'Q96GQ7',
'Q86XP3',
'Q9Y6T7',
'P52429',
'Q9H2U1',
'Q08211',
'Q09013',
'Q16832',
'Q92771',
'Q9NVP1',
'Q9H8H2',
'O15523',
'Q9UJV9',
'Q9Y6V7',
'Q9NQI0',
'Q9NY93',
'O95786',
'Q5KSL6',
'Q13574',
'Q9UPY3',
'Q14565',
```

'Q9UMR2',
'Q08345',
'Q9GZR7',
'Q9UHL0',
'Q9NXZ2',
'Q8NHQ9',
'Q8IY21',
'P27707',
'Q9C098',
'Q9UHI6',
'Q5H9U9',
'O75912',
'O43143',
'Q7Z478',
'Q9H6R0',
'Q96C10',
'Q9H0S4',
'Q16854',
'P00367',
'Q5D0E6',
'Q8N568',
'Q9BUQ8',
'Q9Y2R4',
'Q96FC9',
'Q9NUL7',
'O00571',
'Q8N8A6',
'Q8TDD1',
'Q5T1V6',
'Q86XP1',
'Q9UIK4',
'Q9H5Z1',
'Q8IY37',
'P31689',
'O00148',
'Q13627',
'Q9NR20',
'Q6XUX3',
'Q8TD57',
'Q9C0G6',
'P00533',
'O75417',
'Q96DT5',
'Q96JB1',
'Q9NYC9',
'Q9NZJ5',
'Q92630',

'P49770',
'Q9UFH2',
'Q9P2D7',
'Q8TE73',
'Q8WXX0',
'Q14204',
'Q8NCM8',
'O43781',
'Q9P2K8',
'Q6ZR08',
'Q9P225',
'Q9Y463',
'P49961',
'Q5MY95',
'P54764',
'P29322',
'P54762',
'O15197',
'Q58FF3',
'P22413',
'P21860',
'Q15303',
'P18074',
'Q9NZN3',
'Q96L91',
'Q76MJ5',
'P19525',
'Q9H4M9',
'Q9NZN4',
'Q9H223',
'Q13838',
'Q9BQI3',
'P29317',
'Q5T890',
'O00418',
'Q9HBU6',
'Q9UF33',
'P54753',
'P04626',
'Q2NKX8',
'Q16877',
'Q9NVF9',
'P14625',
'Q9Y5L3',
'P16452',
'Q5JZY3',
'P29323',

'Q03468',
'O75355',
'Q15375',
'O75460',
'P21709',
'P29320',
'P54756',
'P54760',
'P16118',
'Q8IXL6',
'Q6ZS86',
'Q14410',
'Q14409',
'P48637',
'P11021',
'Q8NFF5',
'Q05397',
'Q9BX63',
'P09769',
'A6NMB9',
'P22455',
'Q8NFZ0',
'P11362',
'Q5HY92',
'P19447',
'O60825',
'P16591',
'P22607',
'Q16875',
'Q14296',
'Q14289',
'Q8IYD8',
'Q9BRP7',
'P07332',
'Q02790',
'P42685',
'Q6PIW4',
'P06241',
'Q9Y2I7',
'Q05932',
'P21802',
'Q9BVA6',
'P36888',
'O43716',
'Q8N0W3',
'Q9H0R6',
'Q01415',

```
 'P51570',
 'Q13283',
 'Q5T6J7',
 'Q8IVS8',
 'Q6PIY7',
 'P32189',
 'O14976',
 'Q9NQX3',
 'Q9Y223',
 'P15104',
 'O75879',
 'P49915',
 'Q75WM6',
 'P32298',
 'P49840',
 'Q8WTQ7',
 'P38646',
 'P51841',
 'Q58FF6',
 'P43250',
 'P34947',
 'P49841',
 'P25092',
 'Q58FF8',
 'P48506',
 'Q96GX5',
 'Q8TDG4',
 'Q02846',
 'Q8TF76',
 'Q58FF7',
 'P08631',
 'Q8NG08',
 'Q9BYK8',
 'Q9NRZ9',
 'P42694',
 'Q9H422',
 'Q14527',
 'A2PYH4',
 'Q8NE63',
 …]
```

[39]:
```
max_sequence_size = 500   # any sequence longer than this, we ignore (just for now)
```

[40]:
```
X = []           # sequences in the same order corresponding to elements of p
y = []           # output class: 1 if protein has the function, 0 if not
```

[41]:
```
# for seeing how many examples we've found for each class
pos_examples = 0
neg_examples = 0
```

[42]:
```
with open(sequences_file) as f:
    for line in f:
        ln = line.split(',')
        protein_id = ln[0].strip()
        seq = ln[1].strip()

        # we're doing this to reduce input size
        if len(seq) >= max_sequence_size:
            continue

        print(line)

        X.append(seq)

        if protein_id in has_function:
            y.append(1)
            pos_examples += 1
        else:
            y.append(0)
            neg_examples += 1
```

```
P27361,MAAAAAQGGGGGEPRRTEGVGPGVPGEVEMVKGQPFDVGPRYTQLQYIGEGAYGMVSSAYDHVRKTRVAIKKI
SPFEHQTYCQRTLREIQILLRFRHENVIGIRDILRASTLEAMRDVYIVQDLMETDLYKLLKSQQLSNDHICYFLYQILRG
LKYIHSANVLHRDLKPSNLLINTTCDLKICDFGLARIADPEHDHTGFLTEYVATRWYRAPEIMLNSKGYTKSIDIWSVGC
ILAEMLSNRPIFPGKHYLDQLNHILGILGSPSQEDLNCIINMKARNYLQSLPSKTKVAWAKLFPKSDSKALDLLDRMLTF
NPNKRITVEEALAHPYLEQYYDPTDEPVAEEPFTFAMELDDLPKERLKELIFQETARFQPGVLEAP

P53779,MSLHFLYYCSEPTLDVKIAFCQGFDKQVDVSYIAKHYNMSKSKVDNQFYSVEVGDSTFTVLKRYQNLKPIGSG
AQGIVCAAYDAVLDRNVAIKKLSRPFQNQTHAKRAYRELVLMKCVNHKNIISLLNVFTPQKTLEEFQDVYLVMELMDANL
CQVIQMELDHERMSYLLYQMLCGIKHLHSAGIIHRDLKPSNIVVKSDCTLKILDFGLARTAGTSFMMTPYVVTRYYRAPE
VILGMGYKENVDIWSVGCIMGEMVRHKILFPGRDYIDQWNKVIEQLGTPCPEFMKKLQPTVRNYVENRPKYAGLTFPKLF
PDSLFPADSEHNKLKASQARDLLSKMLVIDPAKRISVDDALQHPYINVWYDPAEVEAPPPQIYDKQLDEREHTIEEWKEL
IYKEVMNSEEKTKNGVVKGQPSPSGAAVNSSESLPPSSSVNDISSMSTDQTLASDTDSSLEASAGPLGCCR

Q15049,MTQEPFREELAYDRMPTLERGRQDPASYAPDAKPSDLQLSKRLPPCFSHKTWVFSVLMGSCLLVTSGFSLYLG
NVFPAEMDYLRCAAGSCIPSAIVSFTVSRRNANVIPNFQILFVSTFAVTTTCLIWFGCKLVLNPSAININFNLILLLLLE
LLMAATVIIAARSSEEDCKKKKGSMSDSANILDEVPFPARVLKSYSVVEVIAGISAVLGGIIALNVDDSVSGPHLSVTFF
WILVACFPSAIASHVAAECPSKCLVEVLIAISSLTSPLLFTASGYLSFSIMRIVEMFKDYPPAIKPSYDVLLLLLLLVLL
LQAGLNTGTAIQCVRFKVSARLQGASWDTQNGPQERLAGEVARSPLKEFDKEKAWRAVVVQMAQ

PODMTO,MTGKNWILISTTTPKSLEDEIVGRLLKILFVIFVDLISIIYVVITS

Q7L9L4,MSFLFGSRSSKTFKPKKNIPEGSHQYELLKHAEATLGSGNLRMAVMLPEGEDLNEWVAVNTVDFFNQINMLYG
```

```
TITDFCTEESCPVMSAGPKYEYHWADGTNIKKPIKCSAPKYIDYLMTWVQDQLDDETLFPSKIGVPFPKNFMSVAKTILK
RLFRVYAHIYHQHFDPVIQLQEEAHLNTSFKHFIFFVQEFNLIDRRELAPLQELIEKLTSKDR

Q86TA1,MSIALKQVFNKDKTFRPKRKFEPGTQRFELHKRAQASLNSGVDLKAAVQLPSGEDQNDWVAVHVVDFFNRINL
IYGTICEFCTERTCPVMSGGPKYEYRWQDDLKYKKPTALPAPQYMNLLMDWIEVQINNEEIFPTCVGVPFPKNFLQICKK
ILCRLFRVFVHVYIHHFDRVIVMGAEAHVNTCYKHFYYFVTEMNLIDRKELEPLKEMTSRMCH

Q86TA1,MSIALKQVFNKDKTFRPKRKFEPGTQRFELHKRAQASLNSGVDLKAAVQLPSGEDQNDWVAVHVVDFFNRINL
IYGTICEFCTERTCPVMSGGPKYEYRWQDDLKYKKPTALPAPQYMNLLMDWIEVQINNEEIFPTCVGVPFPKNFLQICKK
ILCRLFRVFVHVYIHHFDRVIVMGAEAHVNTCYKHFYYFVTEMNLIDRKELEPLKEMTSRMCH
```

[43]:
```
print("Positive Examples: %d" % pos_examples)
print("Negative Examples: %d" % neg_examples)  # Total is different because we␣
↪ignored longer sequences
```

```
Positive Examples: 2
Negative Examples: 5
```

[44]:
```
def sequence_to_indices(sequence):
    """Convert amino acid letters to indices.
       _ means no amino acid (used for padding to accommodate for variable␣
↪length)"""

    try:
        acid_letters = ['_', 'A', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L',␣
↪'M',
                'N', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']

        indices = [acid_letters.index(c) for c in list(sequence)]
        return indices
    except Exception:
        print(sequence)
        raise Exception
```

[45]:
```
sequence_to_indices('AC')  # just testing
```

[45]:
```
[1, 2]
```

[56]:
```
X_all = []
for i in range(len(X)):
    x = sequence_to_indices(X[i])
    X_all.append(x)
```

[55]:
```
X_all = np.array(X_all[0])
y_all = np.array(y)
```

[57]:
```python
print(y[0])
print(X_all[0])
print(len(X_all[0]))
```

```
1
[11, 1, 1, 1, 1, 1, 14, 6, 6, 6, 6, 6, 4, 13, 15, 15, 17, 4, 6, 19, 6, 13, 6,
19, 13, 6, 4, 19, 4, 11, 19, 9, 6, 14, 13, 5, 3, 19, 6, 13, 15, 22, 17, 14, 10,
14, 22, 8, 6, 4, 6, 1, 22, 6, 11, 19, 16, 16, 1, 22, 3, 7, 19, 15, 9, 17, 15,
19, 1, 8, 9, 9, 8, 16, 13, 5, 4, 7, 14, 17, 22, 2, 14, 15, 17, 10, 15, 4, 8, 14,
8, 10, 10, 15, 5, 15, 7, 4, 12, 19, 8, 6, 8, 15, 3, 8, 10, 15, 1, 16, 17, 10, 4,
1, 11, 15, 3, 19, 22, 8, 19, 14, 3, 10, 11, 4, 17, 3, 10, 22, 9, 10, 10, 9, 16,
14, 14, 10, 16, 12, 3, 7, 8, 2, 22, 5, 10, 22, 14, 8, 10, 15, 6, 10, 9, 22, 8,
7, 16, 1, 12, 19, 10, 7, 15, 3, 10, 9, 13, 16, 12, 10, 10, 8, 12, 17, 17, 2, 3,
10, 9, 8, 2, 3, 5, 6, 10, 1, 15, 8, 1, 3, 13, 4, 7, 3, 7, 17, 6, 5, 10, 17, 4,
22, 19, 1, 17, 15, 20, 22, 15, 1, 13, 4, 8, 11, 10, 12, 16, 9, 6, 22, 17, 9, 16,
8, 3, 8, 20, 16, 19, 6, 2, 8, 10, 1, 4, 11, 10, 16, 12, 15, 13, 8, 5, 13, 6, 9,
7, 22, 10, 3, 14, 10, 12, 7, 8, 10, 6, 8, 10, 6, 16, 13, 16, 14, 4, 3, 10, 12,
2, 8, 8, 12, 11, 9, 1, 15, 12, 22, 10, 14, 16, 10, 13, 16, 9, 17, 9, 19, 1, 20,
1, 9, 10, 5, 13, 9, 16, 3, 16, 9, 1, 10, 3, 10, 10, 3, 15, 11, 10, 17, 5, 12,
13, 12, 9, 15, 8, 17, 19, 4, 4, 1, 10, 1, 7, 13, 22, 10, 4, 14, 22, 22, 3, 13,
17, 3, 4, 13, 19, 1, 4, 4, 13, 5, 17, 5, 1, 11, 4, 10, 3, 3, 10, 13, 9, 4, 15,
10, 9, 4, 10, 8, 5, 14, 4, 17, 1, 15, 5, 14, 13, 6, 19, 10, 4, 1, 13]
379
```

[58]:
```python
X_all = sequence.pad_sequences(X_all, maxlen=max_sequence_size)  # to overcome␣
↪the variable length issue
```

[59]:
```python
X_all[0]
```

[59]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0, 11,  1,  1,  1,  1,  1, 14,  6,  6,  6,  6,  6,  4, 13, 15,
       15, 17,  4,  6, 19,  6, 13,  6, 19, 13,  6,  4, 19,  4, 11, 19,  9,
        6, 14, 13,  5,  3, 19,  6, 13, 15, 22, 17, 14, 10, 14, 22,  8,  6,
        4,  6,  1, 22,  6, 11, 19, 16, 16,  1, 22,  3,  7, 19, 15,  9, 17,
       15, 19,  1,  8,  9,  9,  8, 16, 13,  5,  4,  7, 14, 17, 22,  2, 14,
       15, 17, 10, 15,  4,  8, 14,  8, 10, 10, 15,  5, 15,  7,  4, 12, 19,
        8,  6,  8, 15,  3,  8, 10, 15,  1, 16, 17, 10,  4,  1, 11, 15,  3,
       19, 22,  8, 19, 14,  3, 10, 11,  4, 17,  3, 10, 22,  9, 10, 10,  9,
       16, 14, 14, 10, 16, 12,  3,  7,  8,  2, 22,  5, 10, 22, 14,  8, 10,
       15,  6, 10,  9, 22,  8,  7, 16,  1, 12, 19, 10,  7, 15,  3, 10,  9,
       13, 16, 12, 10, 10,  8, 12, 17, 17,  2,  3, 10,  9,  8,  2,  3,  5,
```

```
 6, 10,  1, 15,  8,  1,  3, 13,  4,  7,  3,  7, 17,  6,  5, 10, 17,
 4, 22, 19,  1, 17, 15, 20, 22, 15,  1, 13,  4,  8, 11, 10, 12, 16,
 9,  6, 22, 17,  9, 16,  8,  3,  8, 20, 16, 19,  6,  2,  8, 10,  1,
 4, 11, 10, 16, 12, 15, 13,  8,  5, 13,  6,  9,  7, 22, 10,  3, 14,
10, 12,  7,  8, 10,  6,  8, 10,  6, 16, 13, 16, 14,  4,  3, 10, 12,
 2,  8,  8, 12, 11,  9,  1, 15, 12, 22, 10, 14, 16, 10, 13, 16,  9,
17,  9, 19,  1, 20,  1,  9, 10,  5, 13,  9, 16,  3, 16,  9,  1, 10,
 3, 10, 10,  3, 15, 11, 10, 17,  5, 12, 13, 12,  9, 15,  8, 17, 19,
 4,  4,  1, 10,  1,  7, 13, 22, 10,  4, 14, 22, 22,  3, 13, 17,  3,
 4, 13, 19,  1,  4,  4, 13,  5, 17,  5,  1, 11,  4, 10,  3,  3, 10,
13,  9,  4, 15, 10,  9,  4, 10,  8,  5, 14,  4, 17,  1, 15,  5, 14,
13,  6, 19, 10,  4,  1, 13])
```

1 Now we need to split the data

[60]:
```
print(X_all.shape)  # extremely important that you view this!
print(y_all.shape)  # make sure you are comfortable with shapes!
```

```
(7, 500)
(7,)
```

We'll do a basic shuffle and 66%, 33% split.

[61]:
```
n = X_all.shape[0]  # number of data points
```

[62]:
```
# randomize to shuffle first
randomize = np.arange(n)
np.random.shuffle(randomize)
```

[ ]:
```
randomize
```

[63]:
```
X_all = X_all[randomize]
y_all = y_all[randomize]
```

[64]:
```
test_split = round(n * 2 / 3)
X_train = X_all[:test_split]   # start to (just before) test_split
y_train = y_all[:test_split]
X_test  = X_all[test_split:]   # test_split to end
y_test  = y_all[test_split:]
```

[65]:
```
# Print shapes again
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(5, 500)
(5,)
(2, 500)
(2,)
```

2 The Model

[72]:
```
from tensorflow.keras.layers import Embedding, Input, Dropout, Flatten, Dense,␣
↪Activation
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import SGD
```

[79]:
```
num_amino_acids = 23
embedding_dims = 10
nb_epoch = 20
batch_size = 10
```

[80]:
```
model = Sequential()

model.add(Embedding(num_amino_acids, embedding_dims,␣
↪input_length=max_sequence_size  ))
model.add(Flatten())
model.add(Dense(25, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

[81]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 500, 10)           230

 flatten_1 (Flatten)         (None, 5000)              0

 dense_2 (Dense)             (None, 25)                125025

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 125281 (489.38 KB)
Trainable params: 125281 (489.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[82]:
```
model.compile(loss='binary_crossentropy',
              optimizer=SGD(),
metrics=['accuracy'])
```

[83]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 500, 10)           230

 flatten_1 (Flatten)         (None, 5000)              0

 dense_2 (Dense)             (None, 25)                125025

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 125281 (489.38 KB)
Trainable params: 125281 (489.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[84]:
```
hist = model.fit(X_train, y_train,
                 batch_size = batch_size,
                 epochs = nb_epoch,
                 validation_data = (X_test, y_test),
                 verbose=1)
```

```
Epoch 1/20
1/1 [==============================] - 1s 1s/step - loss: 0.6744 - accuracy:
0.6000 - val_loss: 0.4486 - val_accuracy: 1.0000
Epoch 2/20
1/1 [==============================] - 0s 61ms/step - loss: 0.6741 - accuracy:
0.6000 - val_loss: 0.4495 - val_accuracy: 1.0000
Epoch 3/20
1/1 [==============================] - 0s 60ms/step - loss: 0.6737 - accuracy:
0.6000 - val_loss: 0.4504 - val_accuracy: 1.0000
Epoch 4/20
1/1 [==============================] - 0s 62ms/step - loss: 0.6733 - accuracy:
0.6000 - val_loss: 0.4513 - val_accuracy: 1.0000
Epoch 5/20
1/1 [==============================] - 0s 62ms/step - loss: 0.6730 - accuracy:
0.6000 - val_loss: 0.4522 - val_accuracy: 1.0000
Epoch 6/20
1/1 [==============================] - 0s 63ms/step - loss: 0.6726 - accuracy:
0.6000 - val_loss: 0.4531 - val_accuracy: 1.0000
```

```
Epoch 7/20
1/1 [==============================] - 0s 141ms/step - loss: 0.6723 - accuracy:
0.6000 - val_loss: 0.4539 - val_accuracy: 1.0000
Epoch 8/20
1/1 [==============================] - 0s 79ms/step - loss: 0.6719 - accuracy:
0.6000 - val_loss: 0.4548 - val_accuracy: 1.0000
Epoch 9/20
1/1 [==============================] - 0s 60ms/step - loss: 0.6716 - accuracy:
0.6000 - val_loss: 0.4556 - val_accuracy: 1.0000
Epoch 10/20
1/1 [==============================] - 0s 62ms/step - loss: 0.6713 - accuracy:
0.6000 - val_loss: 0.4564 - val_accuracy: 1.0000
Epoch 11/20
1/1 [==============================] - 0s 61ms/step - loss: 0.6709 - accuracy:
0.6000 - val_loss: 0.4572 - val_accuracy: 1.0000
Epoch 12/20
1/1 [==============================] - 0s 61ms/step - loss: 0.6706 - accuracy:
0.6000 - val_loss: 0.4580 - val_accuracy: 1.0000
Epoch 13/20
1/1 [==============================] - 0s 63ms/step - loss: 0.6703 - accuracy:
0.6000 - val_loss: 0.4587 - val_accuracy: 1.0000
Epoch 14/20
1/1 [==============================] - 0s 77ms/step - loss: 0.6699 - accuracy:
0.6000 - val_loss: 0.4595 - val_accuracy: 1.0000
Epoch 15/20
1/1 [==============================] - 0s 97ms/step - loss: 0.6696 - accuracy:
0.6000 - val_loss: 0.4602 - val_accuracy: 1.0000
Epoch 16/20
1/1 [==============================] - 0s 321ms/step - loss: 0.6693 - accuracy:
0.6000 - val_loss: 0.4609 - val_accuracy: 1.0000
Epoch 17/20
1/1 [==============================] - 0s 129ms/step - loss: 0.6690 - accuracy:
0.6000 - val_loss: 0.4616 - val_accuracy: 1.0000
Epoch 18/20
1/1 [==============================] - 0s 68ms/step - loss: 0.6686 - accuracy:
0.6000 - val_loss: 0.4623 - val_accuracy: 1.0000
Epoch 19/20
1/1 [==============================] - 0s 69ms/step - loss: 0.6683 - accuracy:
0.6000 - val_loss: 0.4630 - val_accuracy: 1.0000
Epoch 20/20
1/1 [==============================] - 0s 67ms/step - loss: 0.6680 - accuracy:
0.6000 - val_loss: 0.4637 - val_accuracy: 1.0000
```

3 Changing to the Functional API

[85]:
```
input = Input(shape=(max_sequence_size,))
```

[86]:
```
embedding = Embedding(num_amino_acids, embedding_dims)(input)
```

[87]:
```
x = Flatten()(embedding)
x = Dense(25, activation='sigmoid')(x)
x = Dense(1)(x)
```

[88]:
```
output = Activation('sigmoid')(x)
```

[89]:
```
model = Model([input], output)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 500)]             0

 embedding_2 (Embedding)     (None, 500, 10)           230

 flatten_2 (Flatten)         (None, 5000)              0

 dense_4 (Dense)             (None, 25)                125025

 dense_5 (Dense)             (None, 1)                 26

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 125281 (489.38 KB)
Trainable params: 125281 (489.38 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[90]:
```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\Bilal\keras\lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

[91]:
```
hist = model.fit(X_train, y_train,
                 batch_size = batch_size,
                 epochs = nb_epoch,
                 validation_data = (X_test, y_test),
                 verbose=1)
```

```
Epoch 1/20
1/1 [==============================] - 1s 1s/step - loss: 0.7016 - accuracy:
0.4000 - val_loss: 0.6654 - val_accuracy: 1.0000
Epoch 2/20
1/1 [==============================] - 0s 75ms/step - loss: 0.6522 - accuracy:
1.0000 - val_loss: 0.6151 - val_accuracy: 1.0000
Epoch 3/20
1/1 [==============================] - 0s 70ms/step - loss: 0.6054 - accuracy:
1.0000 - val_loss: 0.5681 - val_accuracy: 1.0000
Epoch 4/20
1/1 [==============================] - 0s 71ms/step - loss: 0.5604 - accuracy:
1.0000 - val_loss: 0.5249 - val_accuracy: 1.0000
Epoch 5/20
1/1 [==============================] - 0s 58ms/step - loss: 0.5173 - accuracy:
1.0000 - val_loss: 0.4861 - val_accuracy: 1.0000
Epoch 6/20
1/1 [==============================] - 0s 74ms/step - loss: 0.4762 - accuracy:
1.0000 - val_loss: 0.4521 - val_accuracy: 1.0000
Epoch 7/20
1/1 [==============================] - 0s 73ms/step - loss: 0.4372 - accuracy:
1.0000 - val_loss: 0.4229 - val_accuracy: 1.0000
Epoch 8/20
1/1 [==============================] - 0s 82ms/step - loss: 0.4003 - accuracy:
1.0000 - val_loss: 0.3986 - val_accuracy: 1.0000
Epoch 9/20
1/1 [==============================] - 0s 66ms/step - loss: 0.3652 - accuracy:
1.0000 - val_loss: 0.3789 - val_accuracy: 1.0000
Epoch 10/20
1/1 [==============================] - 0s 68ms/step - loss: 0.3320 - accuracy:
1.0000 - val_loss: 0.3635 - val_accuracy: 1.0000
Epoch 11/20
1/1 [==============================] - 0s 64ms/step - loss: 0.3007 - accuracy:
1.0000 - val_loss: 0.3519 - val_accuracy: 1.0000
Epoch 12/20
1/1 [==============================] - 0s 61ms/step - loss: 0.2712 - accuracy:
1.0000 - val_loss: 0.3439 - val_accuracy: 1.0000
Epoch 13/20
1/1 [==============================] - 0s 65ms/step - loss: 0.2438 - accuracy:
1.0000 - val_loss: 0.3389 - val_accuracy: 1.0000
Epoch 14/20
1/1 [==============================] - 0s 62ms/step - loss: 0.2185 - accuracy:
```

```
1.0000 - val_loss: 0.3366 - val_accuracy: 1.0000
Epoch 15/20
1/1 [==============================] - 0s 162ms/step - loss: 0.1954 - accuracy:
1.0000 - val_loss: 0.3367 - val_accuracy: 1.0000
Epoch 16/20
1/1 [==============================] - 0s 70ms/step - loss: 0.1747 - accuracy:
1.0000 - val_loss: 0.3389 - val_accuracy: 1.0000
Epoch 17/20
1/1 [==============================] - 0s 55ms/step - loss: 0.1562 - accuracy:
1.0000 - val_loss: 0.3429 - val_accuracy: 1.0000
Epoch 18/20
1/1 [==============================] - 0s 81ms/step - loss: 0.1400 - accuracy:
1.0000 - val_loss: 0.3484 - val_accuracy: 1.0000
Epoch 19/20
1/1 [==============================] - 0s 73ms/step - loss: 0.1258 - accuracy:
1.0000 - val_loss: 0.3551 - val_accuracy: 1.0000
Epoch 20/20
1/1 [==============================] - 0s 75ms/step - loss: 0.1136 - accuracy:
1.0000 - val_loss: 0.3628 - val_accuracy: 1.0000
```

[92]:
```
hist.history
```

[92]:
```
{'loss': [0.7015990018844604,
  0.652221143245697,
  0.6053702235221863,
  0.5603892207145691,
  0.5173071622848511,
  0.47624245285987854,
  0.43723946809768677,
  0.40025314688682556,
  0.3651968240737915,
  0.3320010304450989,
  0.3006538152694702,
  0.27121108770370483,
  0.24377861618995667,
  0.2184792459011078,
  0.19541902840137482,
  0.17465832829475403,
  0.15619421005249023,
  0.13995471596717834,
  0.12580560147762299,
  0.11356601864099503],
 'accuracy': [0.4000000059604645,
  1.0,
  1.0,
  1.0,
  1.0,
```

```
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0],
'val_loss': [0.6654119491577148,
 0.615116536617279,
 0.5680570006370544,
 0.5249013304710388,
 0.48613178730010986,
 0.4520849585533142,
 0.4229198098182678,
 0.3985978364944458,
 0.37890204787254333,
 0.36348748207092285,
 0.351947546005249,
 0.343869686126709,
 0.33886605501174927,
 0.33658164739608765,
 0.3366903066635132,
 0.3388868570327759,
 0.34287944436073303,
 0.3483852446079254,
 0.35512930154800415,
 0.3628464341163635],
'val_accuracy': [1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0,
```

```
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0,
        1.0]}
```

[ ]: