# Lab09-RNN-Report_st122314

March 25, 2022

# 1 Lab09 Report

## 1.1 st122314

[1]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import unicodedata
import string

def findFiles(path):
    return glob.glob(path)

print(findFiles('data/names/*.txt'))

all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)
```

```
['data/names/French.txt', 'data/names/English.txt', 'data/names/German.txt',
'data/names/Portuguese.txt', 'data/names/Russian.txt',
'data/names/Japanese.txt', 'data/names/Scottish.txt', 'data/names/Spanish.txt',
'data/names/Czech.txt', 'data/names/Irish.txt', 'data/names/Greek.txt',
'data/names/Korean.txt', 'data/names/Vietnamese.txt', 'data/names/Chinese.txt',
'data/names/Polish.txt', 'data/names/Arabic.txt', 'data/names/Dutch.txt',
'data/names/Italian.txt']
```

[2]:
```
# Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a/
↪518232/2809427

def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )

print(unicodeToAscii('Ślusàrski'))
```

Slusarski

[3]:
```python
# Build the category_lines dictionary, a list of names per language

category_lines = {}
all_categories = []

# Read a file and split into lines

def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]

for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines

n_categories = len(all_categories)

# Check that it worked

for c in all_categories[:2]:
    print(c)
    print(category_lines[c])
```

```
French
['Abel', 'Abraham', 'Adam', 'Albert', 'Allard', 'Archambault', 'Armistead',
'Arthur', 'Augustin', 'Babineaux', 'Baudin', 'Beauchene', 'Beaulieu',
'Beaumont', 'Belanger', 'Bellamy', 'Bellerose', 'Belrose', 'Berger', 'Beringer',
'Bernard', 'Bertrand', 'Bisset', 'Bissette', 'Blaise', 'Blanc', 'Blanchet',
'Blanchett', 'Bonfils', 'Bonheur', 'Bonhomme', 'Bonnaire', 'Bonnay', 'Bonner',
'Bonnet', 'Borde', 'Bordelon', 'Bouchard', 'Boucher', 'Brisbois', 'Brodeur',
'Bureau', 'Caron', 'Cavey', 'Chaput', 'Charbonneau', 'Charpentier', 'Charron',
'Chastain', 'Chevalier', 'Chevrolet', 'Cloutier', 'Colbert', 'Comtois',
'Cornett', 'Cote', 'Coupe', 'Courtemanche', 'Cousineau', 'Couture', 'Daniau',
"D'aramitz", 'Daviau', 'David', 'Deforest', 'Degarmo', 'Delacroix', 'De la
fontaine', 'Deniau', 'Deniaud', 'Deniel', 'Denis', 'De sauveterre', 'Deschamps',
'Descoteaux', 'Desjardins', 'Desrochers', 'Desrosiers', 'Dubois', 'Duchamps',
'Dufort', 'Dufour', 'Duguay', 'Dupond', 'Dupont', 'Durand', 'Durant', 'Duval',
'Emile', 'Eustis', 'Fabian', 'Fabre', 'Fabron', 'Faucher', 'Faucheux', 'Faure',
'Favager', 'Favre', 'Favreau', 'Fay', 'Felix', 'Firmin', 'Fontaine', 'Forest',
'Forestier', 'Fortier', 'Foss', 'Fournier', 'Gage', 'Gagne', 'Gagnier',
'Gagnon', 'Garcon', 'Gardinier', 'Germain', 'Geroux', 'Giles', 'Girard',
'Giroux', 'Glaisyer', 'Gosse', 'Gosselin', 'Granger', 'Guerin', 'Guillory',
'Hardy', 'Harman', 'Hebert', 'Herbert', 'Herriot', 'Jacques', 'Janvier',
'Jordan', 'Joubert', 'Labelle', 'Lachance', 'Lachapelle', 'Lamar', 'Lambert',
```

'Lane', 'Langlais', 'Langlois', 'Lapointe', 'Larue', 'Laurent', 'Lavigne', 'Lavoie', 'Leandres', 'Lebeau', 'Leblanc', 'Leclair', 'Leclerc', 'Lecuyer', 'Lefebvre', 'Lefevre', 'Lefurgey', 'Legrand', 'Lemaire', 'Lemieux', 'Leon', 'Leroy', 'Lesauvage', 'Lestrange', 'Leveque', 'Levesque', 'Linville', 'Lyon', 'Lyon', 'Macon', 'Marchand', 'Marie', 'Marion', 'Martel', 'Martel', 'Martin', 'Masson', 'Masson', 'Mathieu', 'Mercier', 'Merle', 'Michaud', 'Michel', 'Monet', 'Monette', 'Montagne', 'Moreau', 'Moulin', 'Mullins', 'Noel', 'Oliver', 'Olivier', 'Page', 'Paget', 'Palomer', 'Pan', 'Pape', 'Paquet', 'Paquet', 'Parent', 'Paris', 'Parris', 'Pascal', 'Patenaude', 'Paternoster', 'Paul', 'Pelletier', 'Perrault', 'Perreault', 'Perrot', 'Petit', 'Pettigrew', 'Pierre', 'Plamondon', 'Plourde', 'Poingdestre', 'Poirier', 'Porcher', 'Poulin', 'Proulx', 'Renaud', 'Rey', 'Reyer', 'Richard', 'Richelieu', 'Robert', 'Roche', 'Rome', 'Romilly', 'Rose', 'Rousseau', 'Roux', 'Roy', 'Royer', 'Salomon', 'Salvage', 'Samson', 'Samuel', 'Sargent', 'Sarkozi', 'Sarkozy', 'Sartre', 'Sault', 'Sauvage', 'Sauvageau', 'Sauvageon', 'Sauvageot', 'Sauveterre', 'Savatier', 'Segal', 'Sergeant', 'Severin', 'Simon', 'Solomon', 'Soucy', 'St martin', 'St pierre', 'Tailler', 'Tasse', 'Thayer', 'Thibault', 'Thomas', 'Tobias', 'Tolbert', 'Traver', 'Travere', 'Travers', 'Traverse', 'Travert', 'Tremblay', 'Tremble', 'Victor', 'Victors', 'Villeneuve', 'Vincent', 'Vipond', 'Voclain', 'Yount']
English
['Abbas', 'Abbey', 'Abbott', 'Abdi', 'Abel', 'Abraham', 'Abrahams', 'Abrams', 'Ackary', 'Ackroyd', 'Acton', 'Adair', 'Adam', 'Adams', 'Adamson', 'Adanet', 'Addams', 'Adderley', 'Addinall', 'Addis', 'Addison', 'Addley', 'Aderson', 'Adey', 'Adkins', 'Adlam', 'Adler', 'Adrol', 'Adsett', 'Agar', 'Ahern', 'Aherne', 'Ahmad', 'Ahmed', 'Aikman', 'Ainley', 'Ainsworth', 'Aird', 'Airey', 'Aitchison', 'Aitken', 'Akhtar', 'Akram', 'Alam', 'Alanson', 'Alber', 'Albert', 'Albrighton', 'Albutt', 'Alcock', 'Alden', 'Alder', 'Aldersley', 'Alderson', 'Aldred', 'Aldren', 'Aldridge', 'Aldworth', 'Alesbury', 'Alexandar', 'Alexander', 'Alexnader', 'Alford', 'Algar', 'Ali', 'Alker', 'Alladee', 'Allam', 'Allan', 'Allard', 'Allaway', 'Allcock', 'Allcott', 'Alldridge', 'Alldritt', 'Allen', 'Allgood', 'Allington', 'Alliott', 'Allison', 'Allkins', 'Allman', 'Allport', 'Allsop', 'Allum', 'Allwood', 'Almond', 'Alpin', 'Alsop', 'Altham', 'Althoff', 'Alves', 'Alvey', 'Alway', 'Ambrose', 'Amesbury', 'Amin', 'Amner', 'Amod', 'Amor', 'Amos', 'Anakin', 'Anderson', 'Andersson', 'Anderton', 'Andrew', 'Andrews', 'Angus', 'Anker', 'Anley', 'Annan', 'Anscombe', 'Ansell', 'Anstee', 'Anthony', 'Antic', 'Anton', 'Antony', 'Antram', 'Anwar', 'Appleby', 'Appleton', 'Appleyard', 'Apsley', 'Arah', 'Archer', 'Ardern', 'Arkins', 'Armer', 'Armitage', 'Armour', 'Armsden', 'Armstrong', 'Arnall', 'Arnett', 'Arnold', 'Arnott', 'Arrowsmith', 'Arscott', 'Arthur', 'Artliff', 'Ashbridge', 'Ashbrook', 'Ashby', 'Ashcroft', 'Ashdown', 'Ashe', 'Asher', 'Ashford', 'Ashley', 'Ashman', 'Ashton', 'Ashurst', 'Ashwell', 'Ashworth', 'Askew', 'Aslam', 'Asom', 'Aspey', 'Aspin', 'Aspinall', 'Astbury', 'Astle', 'Astley', 'Aston', 'Atherley', 'Atherstone', 'Atherton', 'Atkin', 'Atkins', 'Atkinson', 'Attard', 'Atter', 'Atterbury', 'Atterton', 'Attewell', 'Attrill', 'Attwood', 'Auberton', 'Auborn', 'Aubrey', 'Austen', 'Austin', 'Auton', 'Avenue', 'Avery', 'Aves', 'Avis', 'Awad', 'Axon', 'Aylett', 'Ayley', 'Ayliffe', 'Ayling', 'Aylott', 'Aylward', 'Ayres', 'Ayton', 'Aziz', 'Bacon', 'Bailey', 'Bain', 'Bainbridge', 'Baines',

'Bains', 'Baird', 'Baker', 'Baldwin', 'Bale', 'Ball', 'Ballantyne', 'Ballard',
'Bamford', 'Bancroft', 'Banks', 'Banner', 'Bannister', 'Barber', 'Barclay',
'Barker', 'Barlow', 'Barnard', 'Barnes', 'Barnett', 'Baron', 'Barr', 'Barrett',
'Barron', 'Barrow', 'Barry', 'Bartlett', 'Barton', 'Bass', 'Bassett',
'Batchelor', 'Bate', 'Bateman', 'Bates', 'Batt', 'Batten', 'Batty', 'Baxter',
'Bayliss', 'Beadle', 'Beal', 'Beale', 'Beamish', 'Bean', 'Bear', 'Beattie',
'Beatty', 'Beaumont', 'Beck', 'Bedford', 'Beech', 'Beer', 'Begum', 'Bell',
'Bellamy', 'Benfield', 'Benjamin', 'Bennett', 'Benson', 'Bentley', 'Berger',
'Bernard', 'Berry', 'Best', 'Bethell', 'Betts', 'Bevan', 'Beveridge', 'Bickley',
'Biddle', 'Biggs', 'Bill', 'Bing', 'Bingham', 'Binnington', 'Birch', 'Bird',
'Bishop', 'Bithell', 'Black', 'Blackburn', 'Blackman', 'Blackmore', 'Blackwell',
'Blair', 'Blake', 'Blakeley', 'Blakey', 'Blanchard', 'Bland', 'Bloggs', 'Bloom',
'Blundell', 'Blythe', 'Bob', 'Boden', 'Boland', 'Bolton', 'Bond', 'Bone',
'Bonner', 'Boon', 'Booth', 'Borland', 'Bostock', 'Boulton', 'Bourne', 'Bouvet',
'Bowden', 'Bowen', 'Bower', 'Bowers', 'Bowes', 'Bowler', 'Bowles', 'Bowman',
'Boyce', 'Boyd', 'Boyle', 'Bracey', 'Bradbury', 'Bradley', 'Bradshaw', 'Brady',
'Brain', 'Braithwaite', 'Bramley', 'Brandrick', 'Bray', 'Breen', 'Brelsford',
'Brennan', 'Brett', 'Brewer', 'Bridges', 'Briggs', 'Bright', 'Bristow',
'Britton', 'Broadbent', 'Broadhurst', 'Broadley', 'Brock', 'Brook', 'Brooke',
'Brooker', 'Brookes', 'Brookfield', 'Brooks', 'Broomfield', 'Broughton',
'Brown', 'Browne', 'Browning', 'Bruce', 'Brunet', 'Brunton', 'Bryan', 'Bryant',
'Bryson', 'Buchan', 'Buchanan', 'Buck', 'Buckingham', 'Buckley', 'Budd', 'Bugg',
'Bull', 'Bullock', 'Burch', 'Burden', 'Burdett', 'Burford', 'Burge', 'Burgess',
'Burke', 'Burland', 'Burman', 'Burn', 'Burnett', 'Burns', 'Burr', 'Burrows',
'Burt', 'Burton', 'Busby', 'Bush', 'Butcher', 'Butler', 'Butt', 'Butter',
'Butterworth', 'Button', 'Buxton', 'Byrne', 'Caddy', 'Cadman', 'Cahill', 'Cain',
'Cairns', 'Caldwell', 'Callaghan', 'Callow', 'Calveley', 'Calvert', 'Cameron',
'Campbell', 'Cann', 'Cannon', 'Caplan', 'Capper', 'Carey', 'Carling',
'Carmichael', 'Carnegie', 'Carney', 'Carpenter', 'Carr', 'Carrington',
'Carroll', 'Carruthers', 'Carson', 'Carter', 'Cartwright', 'Carty', 'Casey',
'Cashmore', 'Cassidy', 'Caton', 'Cavanagh', 'Cawley', 'Chadwick', 'Chalmers',
'Chamberlain', 'Chambers', 'Chan', 'Chance', 'Chandler', 'Chantler', 'Chaplin',
'Chapman', 'Chappell', 'Chapple', 'Charge', 'Charles', 'Charlton', 'Charnock',
'Chase', 'Chatterton', 'Chauhan', 'Cheetham', 'Chelmy', 'Cherry', 'Cheshire',
'Chester', 'Cheung', 'Chidlow', 'Child', 'Childs', 'Chilvers', 'Chisholm',
'Chong', 'Christie', 'Christy', 'Chung', 'Church', 'Churchill', 'Clamp',
'Clancy', 'Clark', 'Clarke', 'Clarkson', 'Clay', 'Clayton', 'Cleary', 'Cleaver',
'Clegg', 'Clements', 'Cliff', 'Clifford', 'Clifton', 'Close', 'Clough',
'Clowes', 'Coates', 'Coburn', 'Cochrane', 'Cockburn', 'Cockle', 'Coffey',
'Cohen', 'Cole', 'Coleman', 'Coles', 'Coll', 'Collard', 'Collett', 'Colley',
'Collier', 'Collingwood', 'Collins', 'Collinson', 'Colman', 'Compton',
'Conneely', 'Connell', 'Connelly', 'Connolly', 'Connor', 'Conrad', 'Conroy',
'Conway', 'Cook', 'Cooke', 'Cookson', 'Coomber', 'Coombes', 'Cooper', 'Cope',
'Copeland', 'Copland', 'Copley', 'Corbett', 'Corcoran', 'Core', 'Corlett',
'Cormack', 'Corner', 'Cornish', 'Cornock', 'Corr', 'Corrigan', 'Cosgrove',
'Costa', 'Costello', 'Cotter', 'Cotterill', 'Cotton', 'Cottrell', 'Couch',
'Coulson', 'Coulter', 'Court', 'Cousin', 'Cousins', 'Cove', 'Cowan', 'Coward',
'Cowell', 'Cowie', 'Cowley', 'Cox', 'Coyle', 'Crabb', 'Crabtree', 'Cracknell',

'Craig', 'Crane', 'Craven', 'Crawford', 'Crawley', 'Creasey', 'Cresswell',
'Crew', 'Cripps', 'Crisp', 'Crocker', 'Croft', 'Crofts', 'Cronin', 'Crook',
'Crosby', 'Cross', 'Crossland', 'Crossley', 'Crouch', 'Croucher', 'Crow',
'Crowe', 'Crowley', 'Crown', 'Crowther', 'Crump', 'Cullen', 'Cumming',
'Cummings', 'Cummins', 'Cunningham', 'Curley', 'Curran', 'Currie', 'Curry',
'Curtis', 'Curwood', 'Cutts', 'D arcy', 'Dacey', 'Dack', 'Dalby', 'Dale',
'Daley', 'Dallas', 'Dalton', 'Daly', 'Dalzell', 'Damon', 'Danby', 'Dandy',
'Daniel', 'Daniells', 'Daniels', 'Danks', 'Dann', 'Darby', 'Darbyshire',
'Darcy', 'Dardenne', 'Darlington', 'Darr', 'Daugherty', 'Davenport', 'Davey',
'David', 'Davidson', 'Davie', 'Davies', 'Davis', 'Davison', 'Davy', 'Dawe',
'Dawes', 'Dawkins', 'Dawson', 'Day', 'Dayman', 'De ath', 'Deacon', 'Deakin',
'Dean', 'Deane', 'Deans', 'Debenham', 'Deegan', 'Deeley', 'Deighton',
'Delamarre', 'Delaney', 'Dell', 'Dempsey', 'Dempster', 'Denby', 'Denham',
'Denis', 'Denney', 'Dennis', 'Dent', 'Denton', 'Depp', 'Dermody', 'Derrick',
'Derrien', 'Dervish', 'Desai', 'Devaney', 'Devenish', 'Deverell', 'Devine',
'Devlin', 'Devon', 'Devonport', 'Dewar', 'Dexter', 'Diamond', 'Dibble', 'Dick',
'Dickens', 'Dickenson', 'Dicker', 'Dickinson', 'Dickson', 'Dillon', 'Dimmock',
'Dingle', 'Dipper', 'Dixon', 'Dobbin', 'Dobbins', 'Doble', 'Dobson', 'Docherty',
'Docker', 'Dodd', 'Dodds', 'Dodson', 'Doherty', 'Dolan', 'Dolcy', 'Dolman',
'Dolton', 'Donald', 'Donaldson', 'Donkin', 'Donlan', 'Donn', 'Donnachie',
'Donnelly', 'Donoghue', 'Donohoe', 'Donovan', 'Dooley', 'Doolin', 'Doon',
'Doors', 'Dora', 'Doran', 'Dorman', 'Dornan', 'Dorrian', 'Dorrington', 'Dougal',
'Dougherty', 'Doughty', 'Douglas', 'Douthwaite', 'Dove', 'Dover', 'Dowell',
'Dowler', 'Dowling', 'Down', 'Downer', 'Downes', 'Downey', 'Downie', 'Downing',
'Downs', 'Downton', 'Dowson', 'Doyle', 'Drabble', 'Drain', 'Drake', 'Draper',
'Drew', 'Drewett', 'Dreyer', 'Driffield', 'Drinkwater', 'Driscoll', 'Driver',
'Drummond', 'Drury', 'Drysdale', 'Dubois', 'Duck', 'Duckworth', 'Ducon',
'Dudley', 'Duff', 'Duffield', 'Duffin', 'Duffy', 'Dufour', 'Duggan', 'Duke',
'Dukes', 'Dumont', 'Duncan', 'Dundon', 'Dunford', 'Dunkley', 'Dunlop',
'Dunmore', 'Dunn', 'Dunne', 'Dunnett', 'Dunning', 'Dunsford', 'Dupont',
'Durand', 'Durant', 'Durber', 'Durham', 'Durrant', 'Dutt', 'Duval', 'Duvall',
'Dwyer', 'Dyde', 'Dyer', 'Dyerson', 'Dykes', 'Dymond', 'Dymott', 'Dyson',
'Eade', 'Eadie', 'Eagle', 'Eales', 'Ealham', 'Ealy', 'Eames', 'Eansworth',
'Earing', 'Earl', 'Earle', 'Earley', 'Easdale', 'Easdown', 'Easen', 'Eason',
'East', 'Eastaugh', 'Eastaway', 'Eastell', 'Easterbrook', 'Eastham', 'Easton',
'Eastwood', 'Eatherington', 'Eaton', 'Eaves', 'Ebbs', 'Ebden', 'Ebdon',
'Ebeling', 'Eburne', 'Eccles', 'Eccleston', 'Ecclestone', 'Eccott', 'Eckersall',
'Eckersley', 'Eddison', 'Eddleston', 'Eddy', 'Eden', 'Edeson', 'Edgar', 'Edge',
'Edgell', 'Edgerton', 'Edgley', 'Edgson', 'Edkins', 'Edler', 'Edley',
'Edlington', 'Edmond', 'Edmonds', 'Edmondson', 'Edmunds', 'Edmundson', 'Edney',
'Edon', 'Edwards', 'Edwick', 'Eedie', 'Egan', 'Egerton', 'Eggby', 'Eggison',
'Eggleston', 'Eglan', 'Egleton', 'Eglin', 'Eilers', 'Ekin', 'Elbutt', 'Elcock',
'Elder', 'Eldeston', 'Eldridge', 'Eley', 'Elfman', 'Elford', 'Elkin',
'Elkington', 'Ellam', 'Ellans', 'Ellard', 'Elleray', 'Ellerby', 'Ellershaw',
'Ellery', 'Elliman', 'Elling', 'Ellingham', 'Elliot', 'Elliott', 'Ellis',
'Ellison', 'Elliston', 'Ellrott', 'Ellwood', 'Elmer', 'Elmes', 'Elmhirst',
'Elmore', 'Elms', 'Elphick', 'Elsdon', 'Elsmore', 'Elson', 'Elston', 'Elstone',
'Eltis', 'Elven', 'Elvin', 'Elvins', 'Elwell', 'Elwood', 'Elworthy', 'Elzer',

'Emberey', 'Emberson', 'Embleton', 'Emerick', 'Emerson', 'Emery', 'Emmanuel', 'Emmerson', 'Emmery', 'Emmett', 'Emmings', 'Emmins', 'Emmons', 'Emmott', 'Emms', 'Emsden', 'Endroe', 'England', 'English', 'Ennis', 'Ennos', 'Enright', 'Enticott', 'Entwistle', 'Epsom', 'Epton', 'Ernest', 'Erridge', 'Errington', 'Errity', 'Esan', 'Escott', 'Eskins', 'Eslick', 'Espley', 'Essam', 'Essan', 'Essop', 'Estlick', 'Etchells', 'Etheridge', 'Etherington', 'Etherton', 'Ettrick', 'Evans', 'Evason', 'Evenden', 'Everdell', 'Everett', 'Everill', 'Everitt', 'Everson', 'Everton', 'Eveson', 'Evison', 'Evrard', 'Ewart', 'Ewin', 'Ewing', 'Ewles', 'Exley', 'Exon', 'Exton', 'Eyett', 'Eyles', 'Eyre', 'Eyres', 'Fabb', 'Fagan', 'Fagon', 'Fahy', 'Fairbairn', 'Fairbrace', 'Fairbrother', 'Fairchild', 'Fairclough', 'Fairhurst', 'Fairley', 'Fairlie', 'Fairweather', 'Falconer', 'Falk', 'Fall', 'Fallon', 'Fallows', 'Falsh', 'Farge', 'Fargher', 'Farhall', 'Farley', 'Farmer', 'Farnsworth', 'Farnum', 'Farnworth', 'Farr', 'Farrant', 'Farrar', 'Farre', 'Farrell', 'Farrelly', 'Farren', 'Farrer', 'Farrier', 'Farrington', 'Farrow', 'Faulkner', 'Faust', 'Fawcett', 'Fawn', 'Faye', 'Fearn', 'Fearnley', 'Fearns', 'Fearon', 'Featherstone', 'Feeney', 'Feetham', 'Felix', 'Fell', 'Fellmen', 'Fellows', 'Feltham', 'Felton', 'Fenlon', 'Fenn', 'Fenton', 'Fenwick', 'Ferdinand', 'Fereday', 'Ferguson', 'Fern', 'Fernandez', 'Ferns', 'Fernyhough', 'Ferreira', 'Ferrier', 'Ferris', 'Ferry', 'Fewtrell', 'Field', 'Fielder', 'Fielding', 'Fields', 'Fifield', 'Finan', 'Finbow', 'Finch', 'Findlay', 'Findley', 'Finlay', 'Finn', 'Finnegan', 'Finney', 'Finnigan', 'Finnimore', 'Firth', 'Fischer', 'Fish', 'Fisher', 'Fishlock', 'Fisk', 'Fitch', 'Fitchett', 'Fitton', 'Fitzgerald', 'Fitzpatrick', 'Fitzsimmons', 'Flack', 'Flaherty', 'Flanagan', 'Flanders', 'Flannery', 'Flavell', 'Flaxman', 'Fleetwood', 'Fleming', 'Fletcher', 'Flett', 'Florey', 'Floss', 'Flower', 'Flowers', 'Floyd', 'Flynn', 'Foden', 'Fogg', 'Foley', 'Fontaine', 'Foran', 'Forbes', 'Ford', 'Forde', 'Fordham', 'Foreman', 'Forester', 'Forman', 'Forrest', 'Forrester', 'Forshaw', 'Forster', 'Forsyth', 'Forsythe', 'Forth', 'Fortin', 'Foss', 'Fossard', 'Fosse', 'Foster', 'Foston', 'Fothergill', 'Fotheringham', 'Foucher', 'Foulkes', 'Fountain', 'Fowler', 'Fowley', 'Fox', 'Foxall', 'Foxley', 'Frame', 'Frampton', 'France', 'Francis', 'Franco', 'Frankish', 'Frankland', 'Franklin', 'Franks', 'Frary', 'Fraser', 'Frazer', 'Frederick', 'Frederikson', 'Freeburn', 'Freedman', 'Freeman', 'Freestone', 'Freeth', 'Freight', 'French', 'Fretwell', 'Frey', 'Fricker', 'Friel', 'Friend', 'Frith', 'Froggatt', 'Froggett', 'Frost', 'Frostick', 'Froy', 'Frusher', 'Fryer', 'Fulker', 'Fuller', 'Fulleron', 'Fullerton', 'Fulton', 'Funnell', 'Furey', 'Furlong', 'Furnell', 'Furness', 'Furnish', 'Furniss', 'Furse', 'Fyall', 'Gadsden', 'Gaffney', 'Galbraith', 'Gale', 'Gales', 'Gall', 'Gallacher', 'Gallagher', 'Galliford', 'Gallo', 'Galloway', 'Galvin', 'Gamble', 'Gammer', 'Gammon', 'Gander', 'Gandham', 'Ganivet', 'Garber', 'Garbett', 'Garbutt', 'Garcia', 'Gardener', 'Gardiner', 'Gardner', 'Garland', 'Garner', 'Garrard', 'Garratt', 'Garrett', 'Garside', 'Garvey', 'Gascoyne', 'Gaskell', 'Gately', 'Gates', 'Gaudin', 'Gaumont', 'Gauntlett', 'Gavin', 'Gaynor', 'Geaney', 'Geary', 'Geeson', 'Geldard', 'Geldart', 'Gell', 'Gemmell', 'Gene', 'George', 'Gerard', 'Gerrard', 'Geyer', 'Gibb', 'Gibbins', 'Gibbon', 'Gibbons', 'Gibbs', 'Giblin', 'Gibson', 'Gifford', 'Gilbert', 'Gilbey', 'Gilchrist', 'Gilder', 'Giles', 'Gilfillan', 'Gilks', 'Gill', 'Gillam', 'Gillan', 'Gillard', 'Gillen', 'Gillespie', 'Gillett', 'Gillies', 'Gilmartin', 'Gilmore', 'Gilmour',

'Ginty', 'Girdwood', 'Girling', 'Given', 'Gladwell', 'Glaister', 'Glasby',
'Glasgow', 'Glass', 'Gleave', 'Gledhill', 'Gleeson', 'Glen', 'Glencross',
'Glenn', 'Glennie', 'Glennon', 'Glew', 'Glossop', 'Glover', 'Glynn', 'Goble',
'Godby', 'Goddard', 'Godden', 'Godfrey', 'Godwin', 'Goff', 'Gold', 'Goldberg',
'Golding', 'Goldman', 'Goldsmith', 'Goldsworthy', 'Gomez', 'Gonzalez', 'Gooch',
'Good', 'Goodacre', 'Goodall', 'Goodchild', 'Goode', 'Gooding', 'Goodman',
'Goodridge', 'Goodson', 'Goodwin', 'Goodyear', 'Gordon', 'Goring', 'Gorman',
'Gosden', 'Gosling', 'Gough', 'Gould', 'Goulden', 'Goulding', 'Gourlay',
'Govender', 'Govier', 'Gower', 'Gowing', 'Grady', 'Graham', 'Grainger',
'Grange', 'Granger', 'Grant', 'Graves', 'Gray', 'Grayson', 'Greaves', 'Green',
'Greenall', 'Greenaway', 'Greene', 'Greener', 'Greenhill', 'Greening',
'Greenleaf', 'Greenshields', 'Greenslade', 'Greensmith', 'Greenway',
'Greenwood', 'Greer', 'Gregory', 'Greig', 'Grenard', 'Grennan', 'Gresham',
'Grey', 'Grierson', 'Griff', 'Griffin', 'Griffith', 'Griffiths', 'Griggs',
'Grimes', 'Grimshaw', 'Grinham', 'Grivet', 'Grogan', 'Groom', 'Grose',
'Grosvenor', 'Grout', 'Groves', 'Grundy', 'Guest', 'Guilmard', 'Guinard',
'Gulley', 'Gunby', 'Gunn', 'Gunning', 'Gunston', 'Gunter', 'Guthrie',
'Gutteridge', 'Guttridge', 'Hackett', 'Hadden', 'Haddock', 'Hadfield', 'Hagan',
'Haggett', 'Haigh', 'Haine', 'Haines', 'Hale', 'Halford', 'Hall', 'Hallam',
'Hallett', 'Halliday', 'Halliwell', 'Halstead', 'Hamer', 'Hamill', 'Hamilton',
'Hammond', 'Hamnett', 'Hampson', 'Hampton', 'Hancock', 'Hand', 'Handley',
'Hanlon', 'Hannam', 'Hansen', 'Hanson', 'Harden', 'Harding', 'Hardwick',
'Hardy', 'Hargreaves', 'Harker', 'Harkness', 'Harley', 'Harlow', 'Harman',
'Harness', 'Harper', 'Harries', 'Harrington', 'Harris', 'Harrison', 'Harrop',
'Harry', 'Hart', 'Hartley', 'Harvey', 'Harwood', 'Haslam', 'Hassan', 'Hassani',
'Hastings', 'Hatch', 'Hatton', 'Hawes', 'Hawker', 'Hawkes', 'Hawkins',
'Hawkridge', 'Hawley', 'Haworth', 'Hawtin', 'Hayes', 'Haynes', 'Hayward',
'Head', 'Healey', 'Healy', 'Heath', 'Heathcote', 'Heather', 'Heatley', 'Heaton',
'Hedley', 'Hegney', 'Helley', 'Hellier', 'Helm', 'Hemingway', 'Hemmings',
'Henderson', 'Hendry', 'Heneghan', 'Hennessy', 'Henry', 'Hepburn', 'Hepples',
'Herbert', 'Heritage', 'Heron', 'Herron', 'Hetherington', 'Hewitt', 'Hewlett',
'Heywood', 'Hibbert', 'Hickey', 'Hickman', 'Hicks', 'Higgins', 'Higginson',
'Higgs', 'Hill', 'Hills', 'Hilton', 'Hind', 'Hinde', 'Hindle', 'Hindley',
'Hinds', 'Hine', 'Hinton', 'Hirst', 'Hiscocks', 'Hitchcock', 'Hoare', 'Hobbs',
'Hobson', 'Hocking', 'Hodder', 'Hodge', 'Hodges', 'Hodgkins', 'Hodgkinson',
'Hodgson', 'Hodkinson', 'Hodson', 'Hogan', 'Hogg', 'Holden', 'Holder',
'Holding', 'Holdsworth', 'Hole', 'Holgate', 'Holl', 'Holland', 'Hollis',
'Holloway', 'Holman', 'Holmes', 'Holt', 'Homer', 'Hood', 'Hook', 'Hooper',
'Hooton', 'Hope', 'Hopes', 'Hopkins', 'Hopkinson', 'Hopwood', 'Horn', 'Horne',
'Horner', 'Horrocks', 'Horton', 'Hough', 'Houghton', 'Hoult', 'Houlton',
'Houston', 'Howard', 'Howarth', 'Howden', 'Howe', 'Howell', 'Howells', 'Howes',
'Howie', 'Hoyle', 'Hubbard', 'Hudson', 'Huggins', 'Hughes', 'Hull', 'Hulme',
'Hume', 'Humphrey', 'Humphreys', 'Humphries', 'Hunt', 'Hunter', 'Hurley',
'Hurrell', 'Hurst', 'Hussain', 'Hussein', 'Hussey', 'Hutchings', 'Hutchins',
'Hutchinson', 'Hutchison', 'Hutton', 'Hyde', 'Ianson', 'Ibbotson', 'Ibbs',
'Ibrahim', 'Iddon', 'Iggleden', 'Iles', 'Ilett', 'Illing', 'Illingworth',
'Ilsley', 'Impey', 'Imran', 'Ingermann', 'Ingham', 'Ingle', 'Ingleby',
'Ingledew', 'Inglefield', 'Ingles', 'Inglethorpe', 'Ingram', 'Inker', 'Inman',

'Innalls', 'Innes', 'Inson', 'Ireland', 'Ireson', 'Ironman', 'Ironmonger',
'Irvin', 'Irvine', 'Irving', 'Irwin', 'Isaac', 'Isaacs', 'Isbill', 'Isbitt',
'Isgate', 'Isherwod', 'Isherwood', 'Islam', 'Isman', 'Isnard', 'Issac', 'Ivory',
'Izzard', 'Jackman', 'Jacks', 'Jackson', 'Jacob', 'Jacobs', 'Jacobson',
'Jacques', 'Jaffray', 'Jagger', 'Jakeman', 'James', 'Jameson', 'Jamieson',
'Janes', 'Jansen', 'Jardine', 'Jarman', 'Jarram', 'Jarratt', 'Jarrett',
'Jarrold', 'Jarvis', 'Jasper', 'Jebson', 'Jeffcock', 'Jefferies', 'Jeffers',
'Jefferson', 'Jeffery', 'Jefford', 'Jeffrey', 'Jeffreys', 'Jeffries', 'Jeffs',
'Jems', 'Jenas', 'Jenkin', 'Jenkins', 'Jenkinson', 'Jenks', 'Jenkyns', 'Jenner',
'Jennings', 'Jennison', 'Jennson', 'Jensen', 'Jepson', 'Jermy', 'Jerome',
'Jerry', 'Jervis', 'Jesson', 'Jessop', 'Jevons', 'Jewell', 'Jewers', 'Jewett',
'Jewitt', 'Jewkes', 'Jewson', 'Jiggens', 'Jobson', 'Johannson', 'Johansen',
'Johanson', 'John', 'Johns', 'Johnson', 'Johnston', 'Johnstone', 'Jolley',
'Jolly', 'Jonas', 'Jones', 'Jonhson', 'Jopson', 'Jordan', 'Jordison', 'Jordon',
'Joseph', 'Joss', 'Jourdan', 'Jowett', 'Jowitt', 'Joyce', 'Joynson', 'Jubb',
'Judd', 'Judge', 'Jukes', 'Jupp', 'Jury', 'Kacy', 'Kaddour', 'Kamara',
'Kampfner', 'Kane', 'Kanes', 'Kapoor', 'Karim', 'Karne', 'Karras', 'Kassell',
'Kaufman', 'Kaul', 'Kaur', 'Kavanagh', 'Kay', 'Kaye', 'Kayes', 'Keable', 'Keal',
'Kealey', 'Keane', 'Kearney', 'Kearns', 'Kearsley', 'Kearton', 'Keating',
'Keaveney', 'Keay', 'Keeble', 'Keefe', 'Keegan', 'Keelan', 'Keeler', 'Keeley',
'Keeling', 'Keenan', 'Keene', 'Keetley', 'Keffler', 'Kehoe', 'Keighley',
'Keight', 'Keilty', 'Keir', 'Keith', 'Kelk', 'Kell', 'Kelland', 'Kellems',
'Kellie', 'Kelliher', 'Kelly', 'Kelsall', 'Kelsey', 'Kelso', 'Kemp', 'Kempson',
'Kempster', 'Kendall', 'Kendell', 'Kendrick', 'Kenley', 'Kennard', 'Kennedy',
'Kenneford', 'Kennell', 'Kenneth', 'Kennett', 'Kenney', 'Kenning', 'Kenny',
'Kenrick', 'Kensington', 'Kent', 'Kentwood', 'Kenward', 'Kenworthy', 'Kenyon',
'Keogh', 'Kerby', 'Kernick', 'Kerr', 'Kerrell', 'Kerridge', 'Kerrigan',
'Kerrighen', 'Kerrison', 'Kershaw', 'Ketley', 'Kett', 'Kettell', 'Ketteringham',
'Kettlewell', 'Keward', 'Kewley', 'Keys', 'Keyte', 'Keywood', 'Khalid',
'Khalifa', 'Khalil', 'Khan', 'Kibblewhite', 'Kidd', 'Kiddle', 'Kidman',
'Kidner', 'Kiely', 'Kiernan', 'Kilb', 'Kilbee', 'Kilbey', 'Kilbride', 'Kilburn',
'Kilford', 'Kill', 'Killeen', 'Killen', 'Killick', 'Killock', 'Kilminster',
'Kilmurry', 'Kilnan', 'Kilner', 'Kilroy', 'Kilshaw', 'Kimber', 'Kimble',
'Kinch', 'Kinchin', 'Kinder', 'King', 'Kingdon', 'Kinghorn', 'Kingman', 'Kings',
'Kingscott', 'Kingsley', 'Kingston', 'Kinnaird', 'Kinnear', 'Kinnersley',
'Kinniburgh', 'Kinnison', 'Kinrade', 'Kinsella', 'Kinsey', 'Kinsley', 'Kipling',
'Kirby', 'Kirk', 'Kirkbride', 'Kirkbright', 'Kirkby', 'Kirkland', 'Kirkman',
'Kirkpatrick', 'Kirkwood', 'Kirtley', 'Kirwan', 'Kirwin', 'Kitchen', 'Kitchin',
'Kitching', 'Kitson', 'Kitt', 'Klam', 'Klein', 'Knab', 'Knappett', 'Knibb',
'Knigge', 'Knight', 'Knightley', 'Knighton', 'Knights', 'Knott', 'Knowler',
'Knowles', 'Knox', 'Knoxville', 'Knuckles', 'Knutt', 'Koban', 'Kolt', 'Kone',
'Kore', 'Kouma', 'Kram', 'Kreyling', 'Kristensen', 'Kromberg', 'Kruger',
'Kumar', 'Kurian', 'Kurray', 'Kydd', 'Kyle', 'Kysel', 'Labbe', 'Lacey', 'Lacy',
'Laing', 'Laird', 'Lake', 'Lakey', 'Lakin', 'Lamb', 'Lambert', 'Lambton',
'Lame', 'Lamond', 'Lancaster', 'Lander', 'Lane', 'Lang', 'Langdon', 'Lange',
'Langford', 'Langley', 'Langridge', 'Langston', 'Langton', 'Lanham', 'Laraway',
'Large', 'Larkin', 'Larkings', 'Larsen', 'Larsson', 'Last', 'Latham', 'Lathan',
'Lathey', 'Lattimore', 'Laurie', 'Laver', 'Laverick', 'Lavery', 'Lawal',

'Lawler', 'Lawlor', 'Lawn', 'Lawrance', 'Lawrence', 'Lawrie', 'Laws', 'Lawson',
'Lawther', 'Lawton', 'Laycock', 'Layton', 'Le tissier', 'Leach', 'Leadley',
'Leahy', 'Leake', 'Leal', 'Leary', 'Leaver', 'Leck', 'Leckie', 'Ledger', 'Lee',
'Leech', 'Leedham', 'Leek', 'Leeming', 'Lees', 'Leese', 'Leeson', 'Legg',
'Legge', 'Leggett', 'Leigh', 'Leighton', 'Leitch', 'Leith', 'Lendon', 'Lenihan',
'Lennard', 'Lennon', 'Lennox', 'Leonard', 'Leroy', 'Leslie', 'Lester',
'Lethbridge', 'Levann', 'Levett', 'Levin', 'Levine', 'Levy', 'Lewin',
'Lewington', 'Lewins', 'Lewis', 'Lewry', 'Leyland', 'Leys', 'Leyshon',
'Liddell', 'Liddle', 'Lightfoot', 'Lilley', 'Lilly', 'Lilwall', 'Lincoln',
'Lind', 'Linden', 'Lindo', 'Lindop', 'Lindsay', 'Line', 'Lines', 'Linford',
'Ling', 'Linley', 'Linsby', 'Linton', 'Lister', 'Litchfield', 'Little',
'Littlewood', 'Livermore', 'Livingstone', 'Llewellyn', 'Lloyd', 'Loat', 'Lobb',
'Lock', 'Locke', 'Lockett', 'Lockhart', 'Lockie', 'Lockwood', 'Lockyer',
'Lodge', 'Loft', 'Lofthouse', 'Loftus', 'Logan', 'Lohan', 'Lois', 'Lomas',
'Lomax', 'London', 'Long', 'Longhurst', 'Longley', 'Longworth', 'Lonsdale',
'Lopes', 'Lopez', 'Lord', 'Loudon', 'Loughran', 'Louth', 'Lovatt', 'Love',
'Lovegrove', 'Lovell', 'Lovelock', 'Lovett', 'Lovey', 'Lowbridge', 'Lowdon',
'Lowe', 'Lowes', 'Lowis', 'Lowndes', 'Lowrie', 'Lowry', 'Lucas', 'Luce',
'Lucey', 'Luckhurst', 'Ludgrove', 'Ludkin', 'Ludlow', 'Luke', 'Luker', 'Lumb',
'Lumley', 'Lumsden', 'Lunn', 'Lunt', 'Luscombe', 'Luttrell', 'Luxton', 'Lyall',
'Lyes', 'Lyme', 'Lynas', 'Lynch', 'Lynes', 'Lynn', 'Lyon', 'Lyons', 'Mac',
'Macarthur', 'Macaulay', 'Macdonald', 'Mace', 'Macfarlane', 'Macgregor',
'Machin', 'Macintyre', 'Mack', 'Mackay', 'Mackenzie', 'Mackie', 'Maclean',
'Macleod', 'Macmillan', 'Macpherson', 'Macrae', 'Madden', 'Maddocks', 'Magee',
'Maguire', 'Maher', 'Mahoney', 'Main', 'Mair', 'Major', 'Makin', 'Malley',
'Mallinson', 'Malone', 'Maloney', 'Mangnall', 'Mann', 'Manning', 'Mansell',
'Mansfield', 'Manson', 'Markham', 'Marks', 'Marlow', 'Marr', 'Marriott',
'Marsden', 'Marsh', 'Marshall', 'Martin', 'Martinez', 'Martins', 'Mason',
'Masters', 'Mather', 'Mathers', 'Matheson', 'Mathews', 'Matthams', 'Matthews',
'Maughan', 'Mawson', 'Maxwell', 'May', 'Maynard', 'Mcarthur', 'Mcauley',
'Mcavoy', 'Mcbain', 'Mccabe', 'Mccaffrey', 'Mccall', 'Mccallum', 'Mccann',
'Mccarthy', 'Mccartney', 'Mccluskey', 'Mcclymont', 'Mcconnell', 'Mccormack',
'Mccormick', 'Mccourt', 'Mcculloch', 'Mccullough', 'Mcdermott', 'Mcdonagh',
'Mcdonald', 'Mcdonnell', 'Mcdougall', 'Mcelroy', 'Mcewan', 'Mcfadden',
'Mcfarlane', 'Mcgee', 'Mcghee', 'Mcgill', 'Mcginty', 'Mcgowan', 'Mcgrady',
'Mcgrath', 'Mcgregor', 'Mcgrory', 'Mcguinness', 'Mcguire', 'Mcintosh',
'Mcintyre', 'Mckay', 'Mckee', 'Mckenna', 'Mckenzie', 'Mckeown', 'Mckie',
'Mclaren', 'Mclaughlin', 'Mclean', 'Mclellan', 'Mcleod', 'Mcloughlin',
'Mcmahon', 'Mcmanus', 'Mcmillan', 'Mcnally', 'Mcnamara', 'Mcnaught', 'Mcneil',
'Mcneill', 'Mcnulty', 'Mcphail', 'Mcphee', 'Mcpherson', 'Mcrae', 'Mcshane',
'Mctaggart', 'Meadows', 'Meakin', 'Mears', 'Melia', 'Mellor', 'Meredith',
'Merritt', 'Metcalf', 'Metcalfe', 'Michael', 'Michel', 'Middleton', 'Miles',
'Milford', 'Mill', 'Millar', 'Millard', 'Miller', 'Millett', 'Milligan',
'Millington', 'Mills', 'Millward', 'Milne', 'Milner', 'Milward', 'Mistry',
'Mitchell', 'Moffat', 'Mohamed', 'Mohammed', 'Molloy', 'Molyneux', 'Monaghan',
'Montague', 'Montgomery', 'Moody', 'Moon', 'Mooney', 'Moore', 'Moorhouse',
'Moran', 'More', 'Moreno', 'Moreton', 'Morgan', 'Moriarty', 'Morley', 'Moroney',
'Morris', 'Morrison', 'Morrow', 'Mortimer', 'Morton', 'Moseley', 'Moss',

'Mottram', 'Mould', 'Muir', 'Mullen', 'Mulligan', 'Mullins', 'Mundy', 'Munro', 'Murphy', 'Murray', 'Murrell', 'Mustafa', 'Myatt', 'Myers', 'Nair', 'Nairn', 'Nandi', 'Nanson', 'Nanton', 'Napier', 'Napper', 'Nartey', 'Nash', 'Nason', 'Naughton', 'Naumann', 'Nayler', 'Naylor', 'Naysmith', 'Neal', 'Neale', 'Neary', 'Neave', 'Neaverson', 'Nedd', 'Needham', 'Neeson', 'Negros', 'Neighbour', 'Neill', 'Neilsen', 'Neilson', 'Neish', 'Nelmes', 'Nelms', 'Nelson', 'Nemeth', 'Nero', 'Nesbitt', 'Ness', 'Nessbert', 'Nettleton', 'Neville', 'Nevins', 'Nevis', 'Newall', 'Newberry', 'Newbold', 'Newbury', 'Newby', 'Newcombe', 'Newell', 'Newey', 'Newham', 'Newill', 'Newington', 'Newland', 'Newlands', 'Newman', 'Newsham', 'Newsome', 'Newson', 'Newstead', 'Newton', 'Neyland', 'Nichol', 'Nicholas', 'Nicholl', 'Nicholls', 'Nichols', 'Nicholson', 'Nickel', 'Nickolls', 'Nicks', 'Nicol', 'Nicolas', 'Nicoll', 'Nicolson', 'Nield', 'Nielsen', 'Nielson', 'Nightingale', 'Niles', 'Nilsen', 'Nineham', 'Nisbet', 'Nixon', 'Noach', 'Noakes', 'Nobbs', 'Noble', 'Noggins', 'Nokes', 'Nolan', 'Nood', 'Noon', 'Noonan', 'Norbert', 'Norburn', 'Norbury', 'Norcross', 'Nord', 'Norgate', 'Norgrove', 'Norm', 'Norman', 'Normington', 'Norris', 'Norsworthy', 'North', 'Northcott', 'Norton', 'Norville', 'Norwood', 'Notman', 'Nott', 'Nourse', 'Nova', 'Nowak', 'Nowell', 'Noyce', 'Noyes', 'Nugent', 'Number', 'Nunn', 'Nurse', 'Nurton', 'Nutella', 'Nutman', 'Nutt', 'Nuttall', 'Oakes', 'Oakey', 'Oakley', 'Oaks', 'Oakton', 'Oates', 'Oatridge', 'Oatway', 'Obrien', 'Ocallaghan', 'Oconnell', 'Oconnor', 'Odam', 'Oddie', 'Oddy', 'Odea', 'Odell', 'Odling', 'Odonnell', 'Odonoghue', 'Odriscoll', 'Oflynn', 'Ogden', 'Ogilvie', 'Ogilvy', 'Ogrady', 'Ohalloran', 'Ohara', 'Okeefe', 'Okey', 'Okten', 'Olan', 'Oldfield', 'Oldham', 'Olding', 'Oldland', 'Oldroyd', 'Olds', 'Oleary', 'Oliver', 'Olivier', 'Ollerhead', 'Olley', 'Oloughlin', 'Olsen', 'Olson', 'Omalley', 'Oman', 'Oneil', 'Oneill', 'Opayne', 'Openshaw', 'Oram', 'Orbell', 'Orchard', 'Oreilly', 'Oriley', 'Orman', 'Orme', 'Ormiston', 'Ormond', 'Ormsby', 'Ormston', 'Orrell', 'Orritt', 'Orton', 'Orvis', 'Orwin', 'Osborn', 'Osborne', 'Osman', 'Osmond', 'Ostcliffe', 'Ostler', 'Osullivan', 'Oswald', 'Otoole', 'Otten', 'Otter', 'Ottey', 'Ottley', 'Otton', 'Ould', 'Oulton', 'Overall', 'Overett', 'Overfield', 'Overing', 'Overson', 'Overton', 'Owen', 'Owens', 'Owings', 'Oxby', 'Oxenham', 'Oxley', 'Oxtoby', 'Pack', 'Packard', 'Packer', 'Pagan', 'Page', 'Paige', 'Pailing', 'Paine', 'Painter', 'Paisley', 'Palfrey', 'Palfreyman', 'Palin', 'Pallett', 'Palmer', 'Panesar', 'Pankhurst', 'Pannell', 'Parish', 'Park', 'Parker', 'Parkes', 'Parkin', 'Parkins', 'Parkinson', 'Parks', 'Parmar', 'Parnaby', 'Parnell', 'Parr', 'Parratt', 'Parrott', 'Parry', 'Parsons', 'Partington', 'Partlett', 'Partridge', 'Pascoe', 'Pasfield', 'Paskell', 'Passmore', 'Patchett', 'Patel', 'Pateman', 'Paterson', 'Paton', 'Patrick', 'Patten', 'Patterson', 'Pattinson', 'Pattison', 'Patton', 'Paul', 'Pavot', 'Pawson', 'Payne', 'Peace', 'Peach', 'Peacock', 'Peake', 'Peal', 'Peaper', 'Pearce', 'Pears', 'Pearson', 'Peat', 'Peck', 'Pedley', 'Peebles', 'Peel', 'Peers', 'Pegg', 'Peigne', 'Pell', 'Pelling', 'Pemberton', 'Pender', 'Pendlebury', 'Pendleton', 'Penfold', 'Penn', 'Pennell', 'Penney', 'Pennington', 'Percival', 'Pereira', 'Perez', 'Perkin', 'Perkins', 'Perks', 'Perowne', 'Perrett', 'Perrin', 'Perrins', 'Perry', 'Peters', 'Petersen', 'Peterson', 'Petrova', 'Pett', 'Petticrew', 'Peyton', 'Phelan', 'Phelps', 'Philip', 'Philips', 'Phillips', 'Philpott', 'Phipps', 'Phoenix', 'Pick', 'Pickard', 'Pickering', 'Pickersgill', 'Pickett', 'Pickford', 'Pickthall', 'Picot',

```
'Pierce', 'Piercey', 'Pierre', 'Pigott', 'Pike', 'Pilkington', 'Pillay',
'Pinder', 'Pine', 'Pinkney', 'Pinner', 'Pinnock', 'Pinsmail', 'Pipe', 'Piper',
'Pitcher', 'Pitchford', 'Pitt', 'Pitts', 'Plant', 'Plastow', 'Platt', 'Platts',
'Pledger', 'Plouvin', 'Plumb', 'Plummer', 'Pocock', 'Pointer', 'Pole',
'Pollard', 'Pollock', 'Polson', 'Pomeroy', 'Pomphrey', 'Pond', 'Pooke', 'Poole',
'Poon', 'Pope', 'Porter', 'Potter', 'Potts', 'Poulter', 'Poulton', 'Pounder',
'Povey', 'Powell', 'Power', 'Powers', 'Powis', 'Powles', 'Poyser', 'Pratt',
'Preece', 'Prendergast', 'Prentice', 'Prescott', 'Preston', 'Prevost', 'Price',
'Prime', 'Prince', 'Pringle', 'Prior', 'Pritchard', 'Privett', 'Probert',
'Procter', 'Proctor', 'Prosser', 'Provan', 'Pryor', 'Pugh', 'Pullen', 'Purcell',
'Purkis', 'Purnell', 'Purse', 'Purvis', 'Putt', 'Pyle', 'Quigley', 'Quinlivan',
'Quinn', 'Quinnell', 'Quinton', 'Quirk', 'Quirke', 'Rackham', 'Radcliffe',
'Radford', 'Radley', 'Raeburn', 'Rafferty', 'Rahman', 'Raine', 'Rainey',
'Rainford', 'Ralph', 'Ralston', 'Ramm', 'Rampling', 'Ramsay', 'Ramsden',
'Ramsey', 'Rand', 'Randall', 'Randle', 'Ranger', 'Rankin', 'Ranks', 'Rann',
'Ransom', 'Ranson', 'Rapson', 'Rashid', 'Ratcliffe', 'Raval', 'Raven',
'Ravenscroft', 'Rawlings', 'Rawlinson', 'Rawsthorne', 'Raymond', 'Rayner',
'Read', 'Reade', 'Reader', 'Reading', 'Readle', 'Readman', 'Reardon',
'Reasbeck', 'Reay', 'Redden', 'Redding', 'Reddy', 'Redfern', 'Redhead', 'Redin',
'Redman', 'Redmond', 'Redwood', 'Reed', 'Rees', 'Reese', 'Reeve', 'Reeves',
'Regan', 'Regent', 'Rehman', 'Reid', 'Reilly', 'Reisser', 'Render', 'Renna',
'Rennalls', 'Rennie', 'Renshaw', 'Renwick', 'Reveley', 'Reyes', 'Reygan',
'Reynolds', 'Rhoades', 'Rhodes', 'Rhys', 'Ricci', 'Rice', 'Rich', 'Richards',
'Richardson', 'Riches', 'Richman', 'Richmond', 'Richter', 'Rick', 'Rickard',
'Rickards', 'Rickett', 'Ricketts', 'Riddell', 'Riddle', 'Riddler', 'Ridge',
'Ridgway', 'Ridgwell', 'Ridle', 'Ridley', 'Rigby', 'Rigg', 'Rigley', 'Riley',
'Ring', 'Ripley', 'Rippin', 'Riseborough', 'Ritchie', 'Rivers', 'Rixon',
'Roach', 'Robb', 'Robbins', 'Robe', 'Robert', 'Roberts', 'Robertson', 'Robin',
'Robins', 'Robinson', 'Robishaw', 'Robotham', 'Robson', 'Roche', 'Rochford',
'Rockliffe', 'Rodden', 'Roden', 'Rodger', 'Rodgers', 'Rodham', 'Rodrigues',
'Rodriguez', 'Rodwell', 'Roebuck', 'Roff', 'Roffey', 'Rogan', 'Rogers',
'Rogerson', 'Roles', 'Rolfe', 'Rollinson', 'Roman', 'Romans', 'Ronald',
'Ronflard', 'Rook', 'Rooke', 'Roome', 'Rooney', 'Rootham', 'Roper', 'Ropple',
'Roscoe', 'Rose', 'Rosenblatt', 'Rosenbloom', 'Ross', 'Rosser', 'Rossi',
'Rosso', 'Roth', 'Rothery', 'Rothwell', 'Rouse', 'Roussel', 'Rousset',
'Routledge', 'Rowan', 'Rowe', 'Rowland', 'Rowlands', 'Rowley', 'Rowlinson',
'Rowson', 'Royall', 'Royle', 'Rudd', 'Ruff', 'Rugg', 'Rumbold', 'Rumsey',
'Ruscoe', 'Rush', 'Rushbrooke', 'Rushby', 'Rushton', 'Russel', 'Russell',
'Russon', 'Rust', 'Rutherford', 'Rutter', 'Ryan', 'Ryans', 'Rycroft', 'Ryder',
'Sadiq', 'Sadler', 'Said', 'Saleh', 'Salisbury', 'Sallis', 'Salmon', 'Salt',
'Salter', 'Sampson', 'Samuel', 'Samuels', 'Sanchez', 'Sanders', 'Sanderson',
'Sandison', 'Sands', 'Santos', 'Sargent', 'Saunders', 'Savage', 'Saville',
'Sawyer', 'Saxton', 'Sayers', 'Schmid', 'Schmidt', 'Schofield', 'Scott',
'Searle', 'Seddon', 'Seer', 'Selby', 'Sellars', 'Sellers', 'Senior', 'Sewell',
'Sexton', 'Seymour', 'Shackleton', 'Shah', 'Shakespeare', 'Shand', 'Shanks',
'Shannon', 'Sharkey', 'Sharma', 'Sharp', 'Sharpe', 'Sharples', 'Shaughnessy',
'Shaw', 'Shea', 'Shearer', 'Sheehan', 'Sheldon', 'Shelton', 'Shepherd',
'Sheppard', 'Sheridan', 'Sherman', 'Sherriff', 'Sherry', 'Sherwood', 'Shields',
```

'Shipley', 'Short', 'Shotton', 'Showell', 'Shuttleworth', 'Silcock', 'Silva',
'Simmonds', 'Simmons', 'Simms', 'Simon', 'Simons', 'Simpson', 'Sims',
'Sinclair', 'Singh', 'Singleton', 'Sinha', 'Sisson', 'Sissons', 'Skelly',
'Skelton', 'Skinner', 'Skipper', 'Slade', 'Slater', 'Slattery', 'Sloan',
'Slocombe', 'Small', 'Smallwood', 'Smart', 'Smit', 'Smith', 'Smithson',
'Smullen', 'Smyth', 'Smythe', 'Sneddon', 'Snell', 'Snelling', 'Snow', 'Snowden',
'Snowdon', 'Somerville', 'South', 'Southern', 'Southgate', 'Southwick',
'Sparkes', 'Sparrow', 'Spears', 'Speed', 'Speight', 'Spence', 'Spencer',
'Spicer', 'Spiller', 'Spinks', 'Spooner', 'Squire', 'Squires', 'Stacey',
'Stack', 'Staff', 'Stafford', 'Stainton', 'Stamp', 'Stanfield', 'Stanford',
'Stanley', 'Stannard', 'Stanton', 'Stark', 'Steadman', 'Stedman', 'Steel',
'Steele', 'Steer', 'Steere', 'Stenhouse', 'Stephen', 'Stephens', 'Stephenson',
'Sterling', 'Stevens', 'Stevenson', 'Steward', 'Stewart', 'Stock', 'Stocker',
'Stockley', 'Stoddart', 'Stokes', 'Stokoe', 'Stone', 'Stoppard', 'Storer',
'Storey', 'Storr', 'Stott', 'Stout', 'Strachan', 'Strange', 'Street',
'Stretton', 'Strickland', 'Stringer', 'Strong', 'Stroud', 'Stuart', 'Stubbs',
'Stuckey', 'Sturgess', 'Sturrock', 'Styles', 'Sugden', 'Sullivan', 'Summers',
'Sumner', 'Sunderland', 'Sutherland', 'Sutton', 'Swain', 'Swales', 'Swan',
'Swann', 'Swanson', 'Sweeney', 'Sweeting', 'Swift', 'Sykes', 'Sylvester',
'Symes', 'Symonds', 'Taggart', 'Tailor', 'Tait', 'Talbot', 'Tallett', 'Tamber',
'Tang', 'Tanner', 'Tansey', 'Tansley', 'Tappin', 'Tapping', 'Tapscott', 'Tarr',
'Tarrant', 'Tasker', 'Tate', 'Tatlock', 'Tatlow', 'Tatnell', 'Taurel', 'Tayler',
'Taylor', 'Teague', 'Teal', 'Teale', 'Teasdale', 'Tedd', 'Telford', 'Tell',
'Tellis', 'Tempest', 'Templar', 'Temple', 'Templeman', 'Templeton', 'Tennant',
'Terry', 'Thackeray', 'Thackray', 'Thake', 'Thatcher', 'Thelwell', 'Thirlwall',
'Thirlway', 'Thirlwell', 'Thistlethwaite', 'Thom', 'Thomas', 'Thomason',
'Thompson', 'Thoms', 'Thomson', 'Thonon', 'Thorley', 'Thorndyke', 'Thorne',
'Thornes', 'Thornhill', 'Thornley', 'Thornton', 'Thorp', 'Thorpe', 'Thurbon',
'Thurgood', 'Thurling', 'Thurlow', 'Thurman', 'Thurston', 'Tickner', 'Tidmarsh',
'Tierney', 'Till', 'Tillett', 'Tilley', 'Tilson', 'Tilston', 'Timberlake',
'Timmins', 'Timms', 'Timney', 'Timson', 'Tindall', 'Tindell', 'Tinker',
'Tinkler', 'Tinsley', 'Tipping', 'Tippins', 'Tips', 'Tisdall', 'Titmarsh',
'Titmus', 'Titmuss', 'Titterington', 'Toal', 'Tobin', 'Tocher', 'Todd',
'Tohill', 'Toland', 'Tolley', 'Tollis', 'Tolmay', 'Tomas', 'Tombs', 'Tomes',
'Tomkins', 'Tomlin', 'Tomlinson', 'Tompkin', 'Tompkins', 'Toms', 'Tong',
'Tonge', 'Tonks', 'Tonner', 'Toomer', 'Toomey', 'Topham', 'Topley', 'Topliss',
'Topp', 'Torney', 'Torrance', 'Torrens', 'Torres', 'Tosh', 'Totten', 'Toucet',
'Tovar', 'Tovey', 'Towell', 'Towers', 'Towle', 'Townend', 'Towns', 'Townsend',
'Townsley', 'Tozer', 'Trafford', 'Train', 'Trainor', 'Trattles', 'Travers',
'Travill', 'Travis', 'Traynor', 'Treble', 'Trennery', 'Trent', 'Treseder',
'Trevor', 'Trew', 'Trickett', 'Trigg', 'Trimble', 'Trinder', 'Trollope',
'Troon', 'Trotman', 'Trott', 'Trueman', 'Truman', 'Trump', 'Truscott', 'Tuck',
'Tucker', 'Tuckey', 'Tudor', 'Tuffnell', 'Tufnall', 'Tugwell', 'Tully', 'Tunks',
'Tunstall', 'Turford', 'Turke', 'Turkington', 'Turland', 'Turnbull', 'Turner',
'Turney', 'Turnham', 'Turnock', 'Turrell', 'Turton', 'Turvey', 'Tuthill',
'Tuttle', 'Tutton', 'Tweddle', 'Twigg', 'Twiggs', 'Twine', 'Tyler', 'Tyman',
'Tyne', 'Tyrer', 'Tyrrell', 'Uddin', 'Ullman', 'Ullmann', 'Ulyatt', 'Umney',
'Underdown', 'Underhill', 'Underwood', 'Unsworth', 'Unwin', 'Upfield', 'Upjohn',

```
'Upsdell', 'Upson', 'Upton', 'Urwin', 'Utley', 'Utterson', 'Uttley', 'Utton',
'Uttridge', 'Vale', 'Valentine', 'Vallance', 'Vallins', 'Vallory', 'Valmary',
'Vancoller', 'Vane', 'Vann', 'Vanstone', 'Vanwell', 'Vardy', 'Varey', 'Varley',
'Varndell', 'Vass', 'Vaughan', 'Vaughn', 'Veale', 'Veasey', 'Veevers', 'Veitch',
'Velds', 'Venables', 'Ventura', 'Verdon', 'Verell', 'Verney', 'Vernon',
'Vicary', 'Vicens', 'Vickars', 'Vickerman', 'Vickers', 'Vickery', 'Victor',
'Vikers', 'Villiger', 'Villis', 'Vince', 'Vincent', 'Vine', 'Viner', 'Vines',
'Viney', 'Vinicombe', 'Vinny', 'Vinton', 'Virgo', 'Voakes', 'Vockins', 'Vodden',
'Vollans', 'Voyse', 'Vyner', 'Wade', 'Wadham', 'Waghorn', 'Wagstaff', 'Wain',
'Wainwright', 'Waite', 'Wakefield', 'Wakeford', 'Wakeham', 'Wakelin', 'Waldron',
'Wale', 'Wales', 'Walkden', 'Walker', 'Wall', 'Wallace', 'Waller', 'Walling',
'Wallis', 'Walls', 'Walmsley', 'Walpole', 'Walsh', 'Walshe', 'Walter',
'Walters', 'Walton', 'Wane', 'Wang', 'Warburton', 'Warby', 'Ward', 'Warden',
'Wardle', 'Ware', 'Wareing', 'Waring', 'Warn', 'Warner', 'Warren', 'Warriner',
'Warrington', 'Warwick', 'Water', 'Waterfield', 'Waterhouse', 'Wateridge',
'Waterman', 'Waters', 'Waterson', 'Watkins', 'Watkinson', 'Watling', 'Watson',
'Watt', 'Watters', 'Watts', 'Waugh', 'Wears', 'Weasley', 'Weaver', 'Webb',
'Webber', 'Webster', 'Weeks', 'Weir', 'Welch', 'Weldon', 'Weller', 'Wellington',
'Wellman', 'Wells', 'Welsh', 'Welton', 'Were', 'Werner', 'Werrett', 'West',
'Western', 'Westgate', 'Westlake', 'Weston', 'Westwell', 'Westwood', 'Whalley',
'Wharton', 'Wheatcroft', 'Wheatley', 'Wheeldon', 'Wheeler', 'Whelan',
'Whitaker', 'Whitby', 'White', 'Whiteford', 'Whitehead', 'Whitehouse',
'Whitelaw', 'Whiteley', 'Whitfield', 'Whitham', 'Whiting', 'Whitley',
'Whitlock', 'Whitmore', 'Whittaker', 'Whittingham', 'Whittington', 'Whittle',
'Whittley', 'Whitworth', 'Whyte', 'Wickens', 'Wickham', 'Wicks', 'Widdows',
'Widdowson', 'Wiggins', 'Wigley', 'Wilcox', 'Wild', 'Wilde', 'Wildman',
'Wileman', 'Wiles', 'Wilkes', 'Wilkie', 'Wilkin', 'Wilkins', 'Wilkinson',
'Wilks', 'Wilkshire', 'Will', 'Willett', 'Willetts', 'Williams', 'Williamson',
'Willis', 'Wills', 'Willson', 'Wilmot', 'Wilson', 'Wilton', 'Wiltshire',
'Winder', 'Windsor', 'Winfer', 'Winfield', 'Winman', 'Winn', 'Winship',
'Winstanley', 'Winter', 'Wintersgill', 'Winward', 'Wise', 'Wiseman', 'Wither',
'Withers', 'Wolf', 'Wolfe', 'Wolstencroft', 'Wong', 'Wood', 'Woodcock',
'Woodford', 'Woodhall', 'Woodham', 'Woodhams', 'Woodhead', 'Woodhouse',
'Woodland', 'Woodley', 'Woods', 'Woodward', 'Wooldridge', 'Woollard', 'Woolley',
'Woolnough', 'Wootton', 'Worgan', 'Wormald', 'Worrall', 'Worsnop', 'Worth',
'Worthington', 'Wotherspoon', 'Wragg', 'Wraight', 'Wray', 'Wren', 'Wrench',
'Wrenn', 'Wrigglesworth', 'Wright', 'Wrightson', 'Wyatt', 'Wyer', 'Yabsley',
'Yallop', 'Yang', 'Yapp', 'Yard', 'Yardley', 'Yarker', 'Yarlett', 'Yarnall',
'Yarnold', 'Yarwood', 'Yasmin', 'Yates', 'Yeadon', 'Yeardley', 'Yeardsley',
'Yeates', 'Yeatman', 'Yeldon', 'Yeoman', 'Yeomans', 'Yetman', 'Yeung', 'Yoman',
'Yomkins', 'York', 'Yorke', 'Yorston', 'Youlden', 'Young', 'Younge', 'Younis',
'Youssouf', 'Yule', 'Yusuf', 'Zaoui']
```

[4]:
```
print(category_lines['Italian'][:5])
```

```
['Abandonato', 'Abatangelo', 'Abatantuono', 'Abate', 'Abategiovanni']
```

[5]:

```python
# One-hot encoding of a word vocabulary using scikit-learn's OneHotEncoder

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)
print(encoder.fit_transform([['red'], ['green'], ['blue']]))

# One-hot encoding of a word using numpy

import numpy as np

arr = [2, 1, 0]
max = np.max(arr) + 1
print(np.eye(max)[arr])
```

```
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

[6]:

```python
import torch

# Find letter index from all_letters, e.g. "a" -> 0

def letterToIndex(letter):
    return all_letters.find(letter)

# (For demonstration) turn a letter into a <1 x n_letters> tensor

def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# Turn a line into a <line_length x 1 x n_letters> tensor
# (an array of one-hot letter vectors)

def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

print(letterToTensor('J'))
print(lineToTensor('Jones').size())
```

```
tensor([[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., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
torch.Size([5, 1, 57])
```

## 2 The RNN

[7]:
```python
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

[8]:
```python
input = letterToTensor('A')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input, hidden)
output
```

[8]:
```
tensor([[-2.9221, -2.8667, -2.9440, -2.9279, -2.9974, -2.7915, -2.8015, -2.9156,
         -2.8073, -2.8841, -3.0149, -2.9314, -2.9539, -2.8133, -2.8930, -2.8462,
         -2.8499, -2.9025]], grad_fn=<LogSoftmaxBackward0>)
```

[9]:
```python
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

next_hidden = hidden
```

```
for i in range(input.shape[0]):
    output, next_hidden = rnn(input[i], next_hidden)
    print(output)
```

```
tensor([[-2.9221, -2.8667, -2.9440, -2.9279, -2.9974, -2.7915, -2.8015, -2.9156,
         -2.8073, -2.8841, -3.0149, -2.9314, -2.9539, -2.8133, -2.8930, -2.8462,
         -2.8499, -2.9025]], grad_fn=<LogSoftmaxBackward0>)
tensor([[-2.8566, -3.0259, -2.9264, -2.9519, -2.9932, -2.8795, -2.8142, -2.9693,
         -2.9083, -2.8637, -2.9488, -2.8146, -2.9170, -2.8762, -2.9235, -2.8599,
         -2.7312, -2.8126]], grad_fn=<LogSoftmaxBackward0>)
tensor([[-2.9280, -2.9790, -2.9515, -2.7960, -2.9839, -2.7828, -2.7685, -3.0659,
         -2.8904, -2.8941, -2.9057, -2.8776, -2.9043, -2.9171, -2.9550, -2.9407,
         -2.7395, -2.8095]], grad_fn=<LogSoftmaxBackward0>)
tensor([[-2.9031, -2.9485, -2.8484, -2.8379, -2.9047, -2.8357, -2.8272, -2.9306,
         -2.7791, -2.8567, -3.0414, -2.9693, -2.9370, -2.9176, -2.9623, -2.9015,
         -2.7906, -2.8738]], grad_fn=<LogSoftmaxBackward0>)
tensor([[-2.9160, -2.9032, -2.9298, -2.8983, -2.8618, -2.9117, -2.8022, -2.9138,
         -2.8340, -2.8305, -2.9789, -2.8200, -2.9123, -2.9533, -2.9840, -2.8982,
         -2.7480, -2.9665]], grad_fn=<LogSoftmaxBackward0>)
tensor([[-2.9351, -2.9212, -2.8444, -2.8469, -2.8761, -2.8864, -2.8129, -2.9836,
         -2.8357, -2.9360, -2.9415, -2.9561, -2.9437, -2.8644, -2.8517, -2.9282,
         -2.8477, -2.8375]], grad_fn=<LogSoftmaxBackward0>)
```

## 3 Training

[10]:
```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i

print(categoryFromOutput(output))
```

```
('Scottish', 6)
```

[11]:
```
#add a function to get a random element of our training set:

import random

def randomChoice(l):
    # random.randint range is inclusive thus len(l)-1
    return l[random.randint(0, len(l) - 1)]

def randomTrainingExample():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    category_tensor = torch.tensor([all_categories.index(category)],␣
 ↪dtype=torch.long)
```

```
    line_tensor = lineToTensor(line)
    return category, line, category_tensor, line_tensor

for i in range(10):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    print('category =', category, '/ line =', line)
```

```
category = Spanish / line = Gomez
category = Italian / line = Cipriani
category = Italian / line = Gimondi
category = Portuguese / line = Souza
category = Irish / line = John
category = Japanese / line = Takano
category = Dutch / line = Specht
category = German / line = Garber
category = Scottish / line = Smith
category = Scottish / line = Ross
```

[12]:
```
# For the loss function, let's use negative log likelihood:
criterion = nn.NLLLoss()
```

[13]:
```
#Then a function for training on one sequence:

learning_rate = 0.005 # If you set this too high, it might explode. If too low,␣
 ↪it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(- learning_rate * p.grad.data)

    return output, loss.item()
```

[14]:
```
import time
import math

n_iters = 100000
```

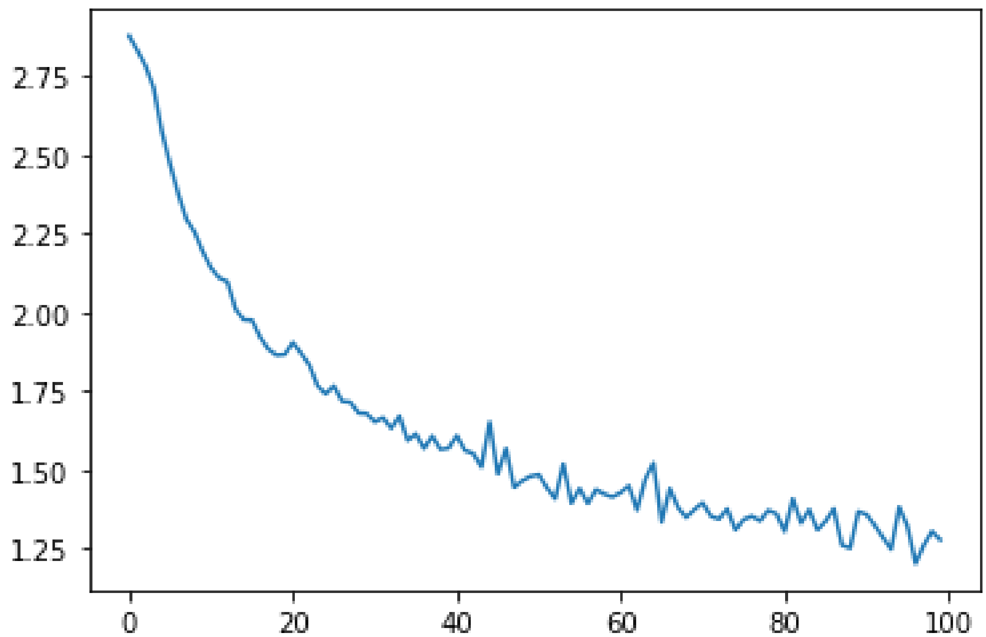
```
print_every = 5000
plot_every = 1000

# Keep track of losses for plotting
current_loss = 0
all_losses = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()

for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = ' ' if guess == category else '  (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100,␣
↪timeSince(start), loss, line, guess, correct))

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

```
5000 5% (0m 18s) 2.8338 Baudin / Scottish   (French)
10000 10% (0m 37s) 2.2187 Callaghan / Russian   (Irish)
15000 15% (0m 55s) 1.0459 Aswad / Arabic
20000 20% (1m 14s) 2.9444 Bishara / Japanese   (Arabic)
25000 25% (1m 32s) 1.5259 Etxeberria / Greek   (Spanish)
30000 30% (1m 51s) 1.0112 Ta / Vietnamese
35000 35% (2m 10s) 0.0344 Rutkowski / Polish
40000 40% (2m 28s) 2.6032 Tombs / Arabic   (English)
45000 45% (2m 47s) 0.9723 Basurto / Portuguese
50000 50% (3m 6s) 2.6317 Cruz / Spanish   (Portuguese)
55000 55% (3m 24s) 1.9146 Bracey / Czech   (English)
60000 60% (3m 42s) 0.1084 Slusarczyk / Polish
65000 65% (4m 0s) 0.0461 Kowalczyk / Polish
70000 70% (4m 18s) 0.3912 Zabek / Polish
```

```
75000 75% (4m 36s) 2.4012 Masson / Scottish  (French)
80000 80% (4m 54s) 0.8807 Maille / Irish
85000 85% (5m 12s) 0.9417 Onoda / Japanese
90000 90% (5m 29s) 0.8131 Wagner / German
95000 95% (5m 48s) 2.1222 Santana / Spanish  (Portuguese)
100000 100% (6m 6s) 2.2920 Grabski / Polish  (Czech)
```

[15]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure()
plt.plot(all_losses)
```

[15]: `[<matplotlib.lines.Line2D at 0x7fd6609c1c10>]`




[16]:
```
all_losses[-1]
```

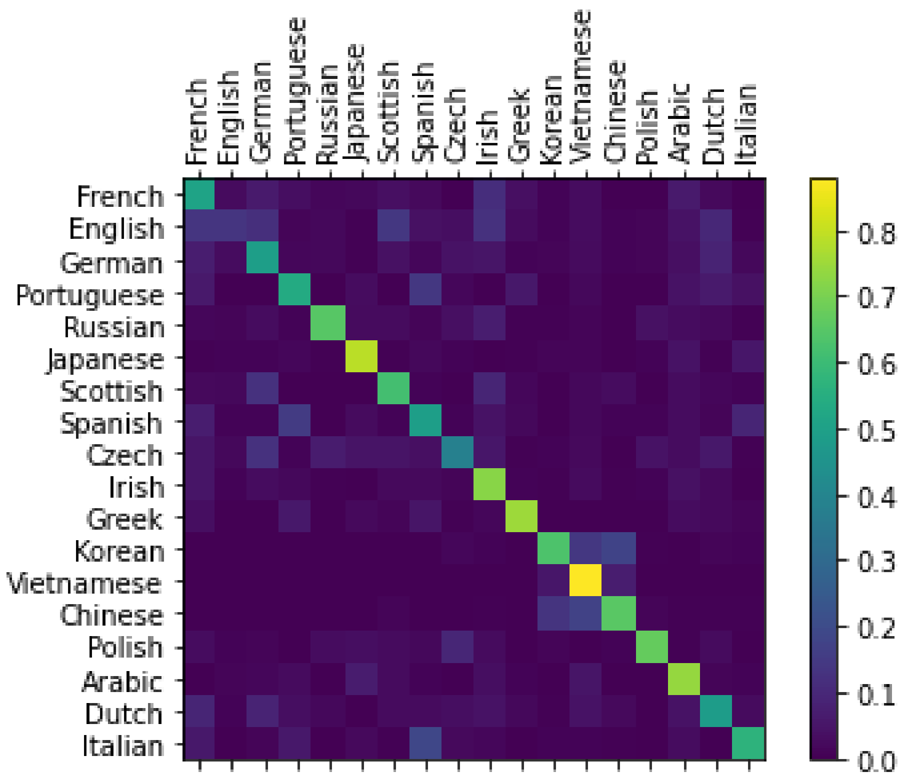
[16]: `1.2768991044315625`

## 4 Evaluation

[17]:
```
# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000
```

```python
# Just return an output given a line
def evaluate(line_tensor):
    hidden = rnn.initHidden()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    return output

# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1

# Normalize by dividing every row by its sum
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()

# Set up plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

# Set up axes
ax.set_xticklabels([''] + all_categories, rotation=90)
ax.set_yticklabels([''] + all_categories)

# Force label at every tick
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# sphinx_gallery_thumbnail_number = 2
plt.show()
```

```
/tmp/ipykernel_22615/3585656379.py:33: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_xticklabels([''] + all_categories, rotation=90)
/tmp/ipykernel_22615/3585656379.py:34: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_yticklabels([''] + all_categories)
```




## 5 Prediction user input

[18]:
```python
def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

        # Get top N categories
        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []

        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('(%.2f) %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])

predict('Ivy')
predict('Phyo')
predict('Kaung')
```

```
> Ivy
(-0.77) Czech
(-2.00) English
(-2.43) Irish

> Phyo
(-0.75) Vietnamese
(-1.96) Korean
(-2.11) Japanese

> Kaung
(-1.49) Chinese
(-1.63) Japanese
(-1.64) German
```

**In this report, I made three parts for independant works which are the following.**

**1. Change the structure to be identical to Goodfellow's Figure 10.3 (no input-to-hidden connection) with tanh activation functions and see if you get different results.**

**2. Explore methods for batching patterns of different length prior to presentation to a RNN and implement them. See how much speedup you can get from the GPU with minibatch training.**

**3. Do a bit of research on similar problems such as named entity recognition, find a dataset, train a model, and report your results.**

# 6 Part1

#### 6.0.1 1. Change the structure to be identical to Goodfellow's Figure 10.3 (no input-to-hidden connection) with tanh activation functions and see if you get different results.

[19]:
```python
import torch.nn as nn

class ElmanRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(ElmanRNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
        self.tanh = nn.Tanh()
```

```
    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        hidden = self.tanh(hidden)
        output = self.i2o(hidden)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
elman_rnn = ElmanRNN(n_letters, n_hidden, n_categories)
```

[20]:
```
print(n_letters)
```

```
57
```

**I employed the Tanh activation layer without input-to-hidden skip connection and the following is the modified the Elman RNN architecture.**

[21]:
```
elman_rnn
```

[21]:
```
ElmanRNN(
  (i2h): Linear(in_features=185, out_features=128, bias=True)
  (i2o): Linear(in_features=128, out_features=18, bias=True)
  (softmax): LogSoftmax(dim=1)
  (tanh): Tanh()
)
```

[22]:
```
def train_Elman_RNN(rnn,category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(- learning_rate * p.grad.data)

    return output, loss.item()
```

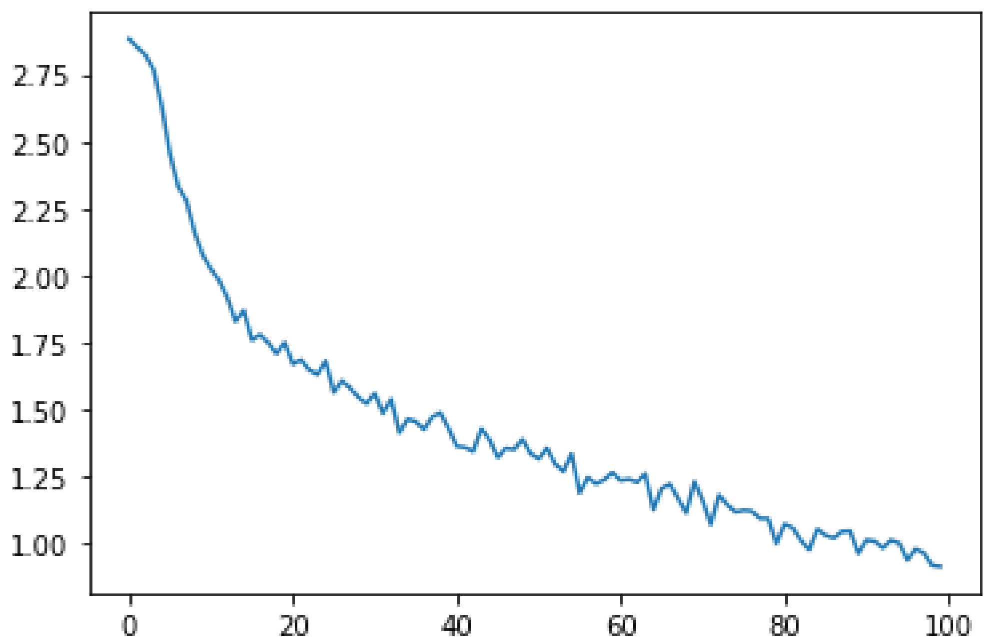
[23]:
```python
import time
import math

n_iters = 100000
print_every = 5000
plot_every = 1000

# Keep track of losses for plotting
current_loss = 0
all_losses = []

# def timeSince(since):
#     now = time.time()
#     s = now - since
#     m = math.floor(s / 60)
#     s -= m * 60
#     return '%dm %ds' % (m, s)

start = time.time()

for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train_Elman_RNN(elman_rnn, category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = ' ' if guess == category else '  (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100,␣
↪timeSince(start), loss, line, guess, correct))

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

```
5000 5% (0m 21s) 2.6220 Tieu / Chinese   (Vietnamese)
10000 10% (0m 49s) 1.5681 Chi / Chinese   (Korean)
15000 15% (1m 20s) 1.4433 Cloutier / French
20000 20% (1m 47s) 1.1147 Sugimura / Japanese
25000 25% (2m 14s) 3.5685 Donk / Korean   (Dutch)
30000 30% (2m 41s) 0.0752 Neroni / Italian
35000 35% (3m 8s) 1.4031 Courtemanche / French
40000 40% (3m 35s) 0.1954 Brzezicki / Polish
45000 45% (4m 1s) 0.6040 Slazak / Polish
50000 50% (4m 23s) 1.1071 Uchida / Japanese
```

```
55000 55% (4m 45s) 0.6956 Romijnsen / Dutch
60000 60% (5m 7s) 0.6070 Bell / Scottish
65000 65% (5m 28s) 3.0695 Kabisha / Japanese  (Russian)
70000 70% (5m 49s) 2.8030 Albuquerque / French  (Portuguese)
75000 75% (6m 10s) 0.2427 Filipek / Polish
80000 80% (6m 31s) 0.2206 Foong / Chinese
85000 85% (6m 51s) 2.9140 Villeneuve / English  (French)
90000 90% (7m 14s) 1.0633 Grosser / German
95000 95% (7m 38s) 0.6649 Chino / Japanese
100000 100% (8m 3s) 1.4134 Flater / Czech  (German)
```

[24]:
```python
import matplotlib.pyplot as plt

plt.figure()
plt.plot(all_losses)
```

[24]: [<matplotlib.lines.Line2D at 0x7fd66147b7f0>]




[25]:
```python
all_losses[-1]
```

[25]: 0.914560677053174

**I got lower losses with the Elman network than using than the simple RNN (1.2768991044315625[simple RNN] vs 0.914560677053174[Elman RNN]). But the training time is also increased (6m 6s[simple RNN] vs 8m 3s[Elman RNN]).**

# 7 Task 2

### 7.0.1 2. Explore methods for batching patterns of different length prior to presentation to a RNN and implement them. See how much speedup you can get from the GPU with minibatch training.

**- batched_lines function will pad all the word with all zero array after each word until the size of that word is equal to the biggest size in the list. Then pack those words into tensor.**

**- batched_categories function is transform tags into tensor.**

[26]:
```
# https://www.marktechpost.com/2020/04/12/
↪implementing-batching-for-seq2seq-models-in-pytorch/
def batched_lines(names, max_word_size):
    rep = torch.zeros(max_word_size, len(names), n_letters)
    for name_index, name in enumerate(names):
        for letter_index, letter in enumerate(name):
            pos = all_letters.find(letter)
            rep[letter_index][name_index][pos] = 1
    return rep

def batched_categories(langs):
    rep = torch.zeros([len(langs)], dtype=torch.long)
    for index, lang in enumerate(langs):
        rep[index] = all_categories.index(lang)
    return rep
```

**For the batching, I implemented to get an array of K random samples from the dataset**

[27]:
```
def randomTrainingBatch(K):
    if(K == 1):
        category = randomChoice(all_categories)
        line = randomChoice(category_lines[category])
        category_tensor = torch.tensor([all_categories.index(category)],␣
↪dtype=torch.long)
        line_tensor = lineToTensor(line)
        return category, line, category_tensor, line_tensor
    else:
        max_length = 0
        categories = []
        lines = []
        lines_length = []
        for i in range(K):
            category = randomChoice(all_categories)
            line = randomChoice(category_lines[category])
            categories.append(category)
            lines.append(line)
            lines_length.append(len(line))
```

```
            if(len(line) > max_length): max_length = len(line)
        line_tensor = batched_lines(lines,max_length)
        category_tensor = batched_categories(categories)
        return categories, lines, category_tensor, line_tensor
```

[28]:
```
randomTrainingBatch(3)
```

[28]:
```
(['Korean', 'Portuguese', 'Polish'],
 ['Yu', 'Cruz', 'Sokolofsky'],
 tensor([11,  3, 14]),
 tensor([[[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.]]]))
```

[29]:
```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
```

```
    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self, batch_size = 1):
        return torch.zeros(batch_size, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

[30]:
```
criterion = nn.NLLLoss()
```

[31]:
```
learning_rate = 0.005 # If you set this too high, it might explode. If too low,␣
 ↪it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden(line_tensor.shape[1])

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(- learning_rate * p.grad.data)

    return output, loss.item()
```

[32]:
```
import time
import math

n_iters = 100000
print_every = 5000
plot_every = 1000

# Keep track of losses for plotting
current_loss = 0
```

```
all_losses = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()


for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingBatch(10)
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = ' ' if guess == category else '  (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100,␣
 ↪timeSince(start), loss, line, guess, correct))


    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

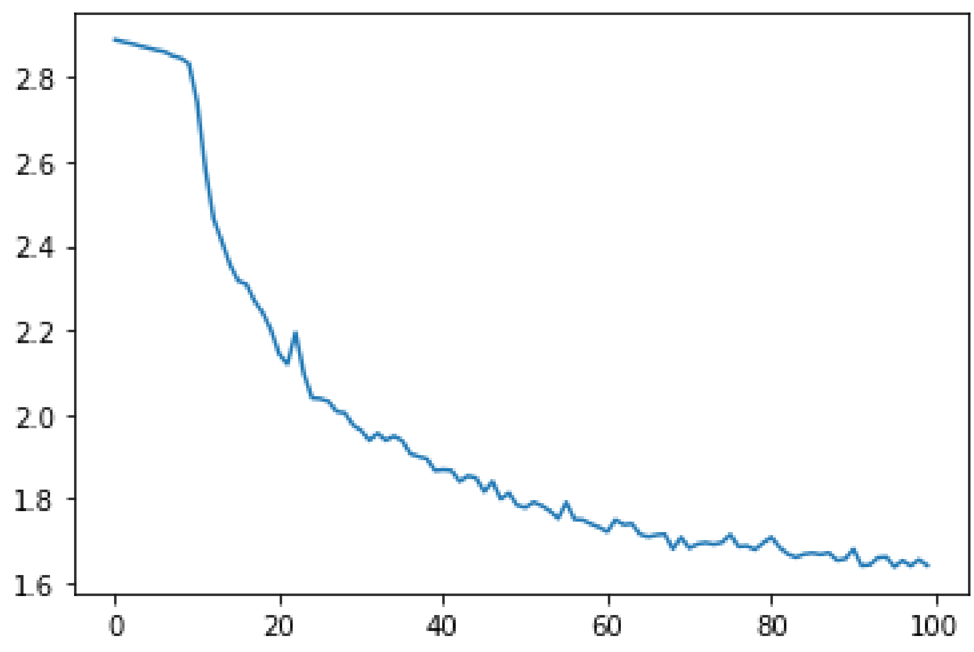
```
5000 5% (0m 20s) 2.8851 ['Asis', 'Nicolson', 'Mata', 'Ly', 'Adsit', 'Fabian',
'Bitar', "O'Sullivan", 'Bernat', 'Pugliese'] / Czech  (['Spanish', 'English',
'Portuguese', 'Vietnamese', 'Czech', 'Polish', 'Arabic', 'Irish', 'German',
'Italian'])
10000 10% (0m 41s) 2.8374 ['Henderson', 'Kowalczyk', 'Laren', 'Van', 'Campos',
'Bakis', 'Flann', 'Rutten', 'Bao', 'Breiner'] / Greek  (['Scottish', 'Polish',
'Dutch', 'Vietnamese', 'Portuguese', 'Russian', 'Irish', 'Dutch', 'Chinese',
'German'])
15000 15% (1m 2s) 2.3693 ['Beringer', 'Walker', 'Pawlitzki', 'Prieto',
'Valerio', 'Emile', 'Mo', 'Peeters', 'Krytinar', 'Kassis'] / German
(['French', 'Scottish', 'German', 'Spanish', 'Italian', 'French', 'Korean',
'Dutch', 'Czech', 'Arabic'])
20000 20% (1m 23s) 2.0906 ['Charron', 'Severijns', 'Hooton', 'Hong', 'Harrison',
'Sala', 'Toomer', 'Kawai', 'Haber', 'Hong'] / Irish  (['French', 'Dutch',
'English', 'Chinese', 'English', 'Spanish', 'English', 'Japanese', 'German',
'Chinese'])
25000 25% (1m 43s) 2.5410 ['Truhanovsky', 'Onohara', 'Collett', 'Noakes',
```

```
'Dymond', 'Zubizarreta', 'Obrien', 'William', 'Dewar', 'Bayer'] / Russian
(['Russian', 'Japanese', 'English', 'English', 'English', 'Spanish', 'English',
'Irish', 'English', 'German'])
30000 30% (2m 3s) 1.6671 ['Suenami', 'Yi', 'Rocca', 'Hofwegen', 'Koury',
'Brown', 'Yim', 'Hynna', 'Laurent', 'Yamanoue'] / Portuguese  (['Japanese',
'Korean', 'Italian', 'Dutch', 'Arabic', 'Scottish', 'Korean', 'Czech', 'French',
'Japanese'])
35000 35% (2m 25s) 2.1295 ['Leigh', 'Koo', "D'cruz", 'Africani', 'Kuiper',
'Rosario', 'Geroux', 'Jiu', 'Faust', 'Pasternak'] / Vietnamese  (['English',
'Korean', 'Portuguese', 'Italian', 'Dutch', 'Portuguese', 'French', 'Chinese',
'English', 'Polish'])
40000 40% (2m 45s) 1.8519 ['Whalen', 'Bereznitzky', 'You', 'Niftrik', 'Halabi',
'Che', 'Marshall', 'Mustafa', 'Sniegowski', 'Prosdocimi'] / German  (['Irish',
'Russian', 'Korean', 'Dutch', 'Arabic', 'Chinese', 'Scottish', 'Arabic',
'Polish', 'Italian'])
45000 45% (3m 4s) 1.8485 ['Vazquez', 'Marchand', 'Fonda', 'Feng', 'Takudo',
'Ozu', 'Seighin', 'Zambrano', 'Gaber', 'Fabre'] / Spanish  (['Spanish',
'French', 'Italian', 'Chinese', 'Japanese', 'Japanese', 'Irish', 'Italian',
'Arabic', 'French'])
50000 50% (3m 23s) 2.0048 ['Phocas', 'Tse', 'Kuai', 'Daly', 'Hawlata', 'Ton',
'Rocha', 'Judd', 'Giolla', 'Gerges'] / Greek  (['Greek', 'Chinese', 'Chinese',
'Irish', 'Czech', 'Vietnamese', 'Portuguese', 'English', 'Irish', 'Arabic'])
55000 55% (3m 43s) 1.3509 ['Drivakis', 'Kneller', 'Sternberg', 'Bustillo',
'Meeuwessen', 'Naser', 'Truong', 'Durant', 'Boyce', 'Ritchie'] / Russian
(['Greek', 'German', 'German', 'Spanish', 'Dutch', 'Arabic', 'Vietnamese',
'French', 'English', 'Scottish'])
60000 60% (4m 3s) 2.3676 ['Rademaker', 'Phocas', 'Haanraats', 'Mar',
'Koutsoubos', 'Maceachthighearna', 'Sakellariou', 'Mcguire', 'Millar',
'Mersinias'] / Dutch  (['Dutch', 'Greek', 'Dutch', 'Chinese', 'Greek', 'Irish',
'Greek', 'Irish', 'Scottish', 'Greek'])
65000 65% (4m 23s) 1.3172 ['Ansaldi', 'Fernandes', 'Joubert', 'Corraidhin',
'Jasso', 'Vantchugov', 'Wojewodzki', 'Ping', 'Ayabito', 'Cho'] / Italian
(['Italian', 'Portuguese', 'French', 'Irish', 'Spanish', 'Russian', 'Polish',
'Chinese', 'Japanese', 'Korean'])
70000 70% (4m 43s) 1.6455 ['Rapson', 'Lac', 'Niemec', 'Huo', "O'Bree", 'Svocak',
'Kalakos', 'Nghiem', 'Romilly', 'Skeril'] / Scottish  (['English',
'Vietnamese', 'Polish', 'Chinese', 'Irish', 'Czech', 'Greek', 'Vietnamese',
'French', 'Czech'])
75000 75% (5m 4s) 1.6626 ['Wood', 'Peeters', "O'Meara", 'Perugia', 'Araujo',
'Sommer', 'Velazquez', 'Reier', 'Kokoris', 'Andringa'] / Korean  (['Czech',
'Dutch', 'Irish', 'Italian', 'Portuguese', 'German', 'Spanish', 'German',
'Greek', 'Dutch'])
80000 80% (5m 25s) 1.7096 ['Bass', 'Kauphsman', 'Tillens', 'Ukiyo', 'Ho',
'Alves', 'Quach', 'Yang', 'Shimaoka', 'Snell'] / Arabic  (['Russian', 'Czech',
'German', 'Japanese', 'Korean', 'Portuguese', 'Vietnamese', 'Korean',
'Japanese', 'English'])
85000 85% (5m 45s) 1.3137 ['Sarumara', 'Castellano', 'Kiski', 'Trang', 'Robert',
'Cathan', 'Medeiros', 'De santigo', 'Ardiccioni', 'Zolotdinov'] / Portuguese
```

```
(['Japanese', 'Spanish', 'Japanese', 'Vietnamese', 'Dutch', 'Irish',
'Portuguese', 'Spanish', 'Italian', 'Russian'])
90000 90% (6m 5s) 1.5103 ['Cernochova', 'Friedrich', 'Manoukarakis', 'Armando',
'Hradek', 'Cham', 'Inihara', 'Mahoney', 'Imai', 'Tsapko'] / Italian  (['Czech',
'German', 'Greek', 'Spanish', 'Czech', 'Arabic', 'Japanese', 'Irish',
'Japanese', 'Russian'])
95000 95% (6m 26s) 1.8288 ['Pahlke', 'Scott', 'Scavo', 'Meisner', 'Penners',
'Fleischer', 'Levitin', 'Czajka', 'Stewart', 'Gasho'] / Czech  (['German',
'Scottish', 'Italian', 'German', 'Dutch', 'German', 'Russian', 'Polish',
'Scottish', 'Russian'])
100000 100% (6m 45s) 1.4194 ['Cunningham', 'La', 'Selmone', 'Benedetti', 'Rios',
'Knochenmus', 'Balashov', 'Otsuka', 'Gutermuth', 'Solos'] / Irish
(['Scottish', 'Vietnamese', 'Italian', 'Italian', 'Portuguese', 'German',
'Russian', 'Japanese', 'German', 'Spanish'])
```

[33]:
```python
import matplotlib.pyplot as plt

plt.figure()
plt.plot(all_losses)
```

[33]: [<matplotlib.lines.Line2D at 0x7fd7d848db50>]




**With minibatch training , it takes faster time than without batching and only 6m 56s.**

# 8 Task 3

#### 8.0.1 3. Do a bit of research on similar problems such as named entity recognition, find a dataset, train a model, and report your results.

**For the named entity recognization dataset, I downloaded from https://www.kaggle.com/datasets/namanj27/ner-dataset.**

[34]:
```
import pandas as pd
data = pd.read_csv('ner_dataset.csv', encoding= 'unicode_escape')
data.head()
```

[34]:
```
    Sentence #           Word  POS Tag
0  Sentence: 1      Thousands  NNS   O
1          NaN             of   IN   O
2          NaN  demonstrators  NNS   O
3          NaN           have  VBP   O
4          NaN        marched  VBN   O
```

[35]:
```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size)
        # A bit more efficient than normal Softmax
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        # print(input.shape, hidden.shape)
        combined = torch.cat((input, hidden), 1)
        a = self.i2h(combined)
        hidden = torch.tanh(a)
        o = self.h2o(hidden)
        y_hat = self.softmax(o)
        # hidden = self.i2h(combined)
        # output = self.i2o(combined)
        # output = self.softmax(output)
        return y_hat, hidden

    def initHidden(self, batch_size = 1):
        return torch.zeros(batch_size, self.hidden_size)
```

[36]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import unicodedata
import string
import torch

# Util Functions
def findFiles(path):
    return glob.glob(path)

def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )

def letterToIndex(letter):
    return all_letters.find(letter)

# (For demonstration) turn a letter into a <1 x n_letters> tensor

def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# Turn a line into a <line_length x 1 x n_letters> tensor
# (an array of one-hot letter vectors)

def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i




# Read a file and split into lines
```

```
def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]


# Prepare Data
all_letters = string.ascii_letters + " .,;'"

category_lines = {}
all_categories = []

n_letters = len(all_letters)

for pos in list(set(data['POS'].to_list())):
    category_lines[pos] = []
    all_categories.append(pos)

for word,pos in zip(data['Word'].to_list(),data['POS'].to_list()):
    category_lines[pos].append(word)


# for filename in findFiles('data/names/*.txt'):
#     category = os.path.splitext(os.path.basename(filename))[0]
#     all_categories.append(category)
#     lines = readLines(filename)
#     category_lines[category] = lines


n_categories = len(all_categories)
```

[37]:

```
# https://www.marktechpost.com/2020/04/12/
↪implementing-batching-for-seq2seq-models-in-pytorch/
def batched_lines(names, max_word_size):
    rep = torch.zeros(max_word_size, len(names), n_letters)
    for name_index, name in enumerate(names):
        for letter_index, letter in enumerate(name):
            pos = all_letters.find(letter)
            rep[letter_index][name_index][pos] = 1
    return rep

def print_char(name_reps):
    # name_reps = name_reps.view((-1, name_reps.size()[-1]))
    # print(name_reps)
    for t in name_reps:
        # if torch.sum(t) == 0:
        #     print('')
        # else:
```

```
            index = t.argmax()
            print(all_letters[index])

def batched_categories(langs):
    rep = torch.zeros([len(langs)], dtype=torch.long)
    for index, lang in enumerate(langs):
        rep[index] = all_categories.index(lang)
    return rep
```

[38]:
```
n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)



import random

def randomChoice(l):
    # random.randint range is inclusive thus len(l)-1
    return l[random.randint(0, len(l) - 1)]

def randomTrainingBatch(batch_size = 1):
    if(batch_size == 1):
        category = randomChoice(all_categories)
        line = randomChoice(category_lines[category])
        category_tensor = torch.tensor([all_categories.index(category)],␣
 ↪dtype=torch.long)
        line_tensor = lineToTensor(line)
        return category, line, category_tensor, line_tensor
    else:
        max_length = 0
        categories = []
        lines = []
        lines_length = []
        for i in range(batch_size):
            category = randomChoice(all_categories)
            line = randomChoice(category_lines[category])
            categories.append(category)
            lines.append(line)
            lines_length.append(len(line))
            if(len(line) > max_length): max_length = len(line)
        line_tensor = batched_lines(lines,max_length)
        category_tensor = batched_categories(categories)
        # padded_line_tensor = torch.nn.utils.rnn.
 ↪pack_padded_sequence(line_tensor, lines_length, enforce_sorted = False)
        return categories, lines, category_tensor, line_tensor

# for i in range(10):
# category, line, category_tensor, line_tensor = randomTrainingExample(10)
```

```
# print(category_tensor.shape, line_tensor.shape)
#     print('category =', category, '/ line =', line)

# If use softmax -> corss entropy
# If use logsoftmax -> negative log likelihood loss
criterion = nn.NLLLoss()
```

[39]:
```
learning_rate = 0.005 # If you set this too high, it might explode. If too low,␣
↪it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden(line_tensor.shape[1])

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item()
```

[40]:
```
import time
import math

n_iters = 100000
print_every = 5000
plot_every = 1000

# Keep track of losses for plotting
current_loss = 0
all_losses = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()
```

```
batch_size = 10
# n_iters = int(n_iters / batch_size)
# print_every = int(print_every / batch_size)
# plot_every = int(plot_every / batch_size)

for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor =␣
↪randomTrainingBatch(batch_size)
    # print(type(category_tensor.to('cuda:1')), type(line_tensor))

    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = ' ' if guess == category else '  (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100,␣
↪timeSince(start), loss, line, guess, correct))

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

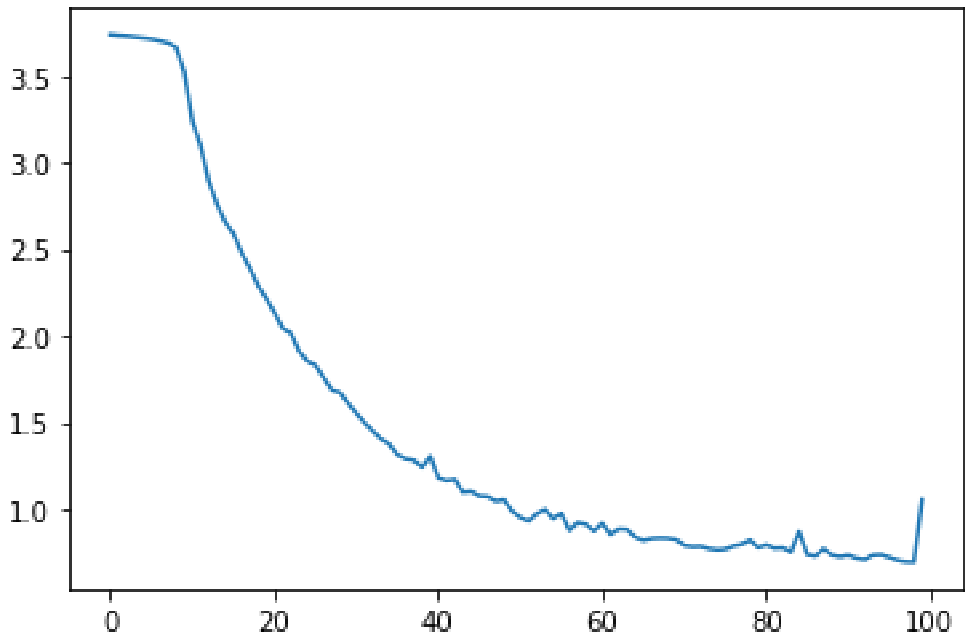
```
/tmp/ipykernel_22615/580995838.py:16: UserWarning: This overload of add_ is
deprecated:
        add_(Number alpha, Tensor other)
Consider using one of the following signatures instead:
        add_(Tensor other, *, Number alpha) (Triggered internally at
../torch/csrc/utils/python_arg_parser.cpp:1050.)
  p.data.add_(-learning_rate, p.grad.data)
```

```
5000 5% (0m 19s) 3.7277 ['positive', 'most', 'Three', 'two', ')', '$', 'where',
'He', 'to', 'further'] / JJR  (['JJ', 'RBS', 'CD', 'CD', 'RRB', '$', 'WRB',
'PRP', 'TO', 'JJ'])
10000 10% (0m 39s) 3.2571 ['pleading', 'earlier', '5', 'more', 'is', 'continue',
')', 'most', 'such', ')'] / VBG  (['VBG', 'JJR', 'CD', 'JJR', 'VBZ', 'VB',
'RRB', 'RBS', 'PDT', 'RRB'])
15000 15% (0m 58s) 2.5040 ['who', '$', 'which', 'can', 'Muslims', 'Monday', ';',
'will', 'his', 'such'] / WP  (['WP', '$', 'WDT', 'MD', 'NNPS', 'NNP', ':',
'MD', 'PRP$', 'PDT'])
20000 20% (1m 17s) 2.1616 ['well', '(', 'earlier', ')', 'Ah', 'that', 'that',
'to', 'their', ')'] / NNPS  (['RB', 'LRB', 'RBR', 'RRB', 'UH', 'WDT', 'WDT',
'TO', 'PRP$', 'RRB'])
25000 25% (1m 36s) 1.8256 ['"', '(', 'are', 'earlier', 'intense', 'and', 'all',
'$', 'more', 'will'] / $  (['``', 'LRB', 'VBP', 'JJR', 'JJ', 'CC', 'PDT', '$',
```

```
'RBR', 'MD'])
30000 30% (1m 54s) 1.8286 ['would', 'Hunters', ']', 'most', 'Kurds', 'is', "'s",
'distribute', 'groups', 'greater'] / WRB  (['MD', 'NNPS', 'RRB', 'RBS', 'NNPS',
'VBZ', 'POS', 'VB', 'NNS', 'JJR'])
35000 35% (2m 14s) 1.8370 ['less', 'and', 'Government', '.', 'States', 'take-
over', 'why', 'killing', 'most', 'measures'] / JJS  (['RBR', 'CC', 'NNP', '.',
'NNPS', 'NN', 'WRB', 'VBG', 'RBS', 'NNS'])
40000 40% (2m 34s) 1.3705 ['(', '"', 'wage', 'such', 'the', 'down', "'s",
'Meanwhile', 'his', '$'] / $  (['LRB', '``', 'NN', 'PDT', 'DT', 'RP', 'POS',
'RB', 'PRP$', '$'])
45000 45% (2m 53s) 1.8636 ['-', '(', 'and', 'said', 'elaborate', 'up', 'are',
'Ah', 'Less', 'determine'] / LRB  ([':', 'LRB', 'CC', 'VBD', 'VB', 'RP', 'VBP',
'UH', 'RBR', 'VB'])
50000 50% (3m 12s) 1.0511 ['least', 'They', '(', 'Tuesday', '1986', '"', 'is',
'whose', 'to', 'come'] / JJS  (['JJS', 'PRP', 'LRB', 'NNP', 'CD', '``', 'VBZ',
'WP$', 'TO', 'VBN'])
55000 55% (3m 31s) 1.1476 ['Oh', 'all', 'one', '.', 'late', 'Rwandan', 'O',
'perestroika', 'Minister', 'opportunity'] / UH  (['UH', 'PDT', 'CD', '.', 'JJ',
'JJ', 'UH', 'FW', 'NNP', 'NN'])
60000 60% (3m 50s) 0.4507 ['recent', 'There', 'Garden', 'favoring', 'Warming',
'in', 'to', 'will', 'was', 'Alas'] / VB  (['JJ', 'EX', 'NNP', 'VBG', 'NNP',
'IN', 'TO', 'MD', 'VBD', 'UH'])
65000 65% (4m 8s) 0.5457 [',', 'separate', 'most', 'or', 'and', '-', 'has',
'called', 'more', 'all'] / ,  ([',', 'JJ', 'RBS', 'CC', 'CC', ':', 'VBZ',
'VBD', 'JJR', 'PDT'])
70000 70% (4m 27s) 0.7163 ['whose', ';', 'raise', 'triggering', 'whose', ')',
'more', 'nearly', '"', 'their'] / WP$  (['WP$', ':', 'VB', 'VBG', 'WP$', 'RRB',
'RBR', 'RB', '``', 'PRP$'])
75000 75% (4m 46s) 1.1284 ['there', 'there', 'and', 'when', 'there', '-', '$',
'northern', 'they', 'leave'] / EX  (['EX', 'EX', 'CC', 'WRB', 'EX', ':', '$',
'JJ', 'PRP', 'VBP'])
80000 80% (5m 6s) 0.9236 ['least', 'perestroika', 'Lords', ')', 'O', 'endorse',
';', 'his', '.', 'have'] / JJS  (['JJS', 'FW', 'NNPS', 'RRB', 'UH', 'VB', ':',
'PRP$', '.', 'VBP'])
85000 85% (5m 26s) 0.6674 ['been', 'Lula', "'s", 'have', 'prices', 'whose', 'O',
'almost', 'succession', 'whose'] / VBN  (['VBN', 'NNP', 'POS', 'VBP', 'NNS',
'WP$', 'UH', 'RB', 'NN', 'WP$'])
90000 90% (5m 45s) 0.5378 ['his', 'to', 'time', '.', '$', 'that', 'can', 'two',
'.', 'higher'] / PRP$  (['PRP$', 'TO', 'NN', '.', '$', 'WDT', 'MD', 'CD', '.',
'JJR'])
95000 95% (6m 4s) 0.1393 ['out', 'perestroika', 'There', 'has', 'of', 'whose',
';', ';', 'will', 'Relations'] / RP  (['RP', 'FW', 'EX', 'VBZ', 'IN', 'WP$',
':', ':', 'MD', 'NNP'])
100000 100% (6m 24s) 2.8905 ['will', 'is', 'least', '-', 'crossed', 'which',
'half', 'are', 'more', 'where'] / MD  (['MD', 'VBZ', 'JJS', ':', 'VBD', 'WDT',
'PDT', 'VBP', 'JJR', 'WRB'])
```

[41]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure()
plt.plot(all_losses)
```

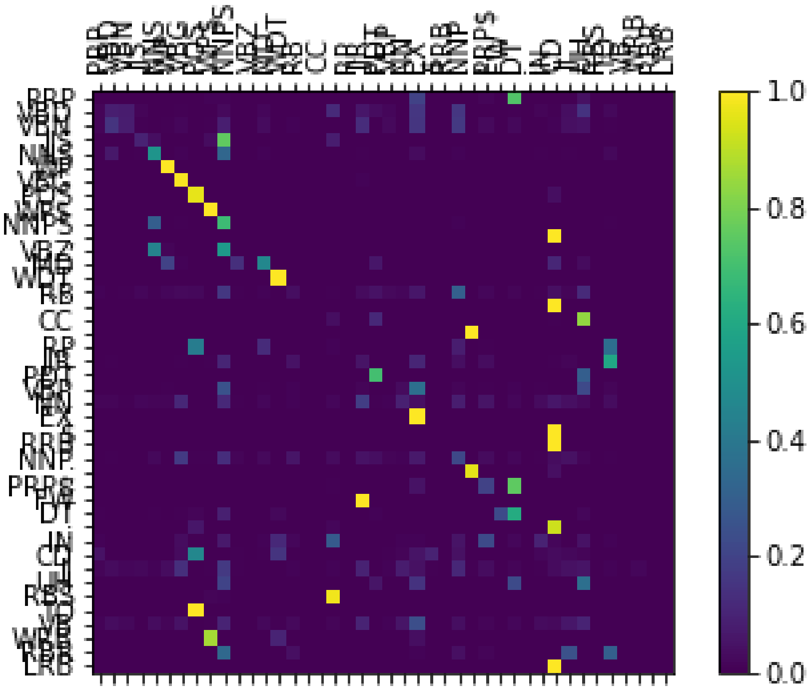
[41]: [<matplotlib.lines.Line2D at 0x7fd66142f520>]




[43]:
```
# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000

# Just return an output given a line
def evaluate(line_tensor):
    hidden = rnn.initHidden()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    return output

# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
```

```
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1

# Normalize by dividing every row by its sum
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()

# Set up plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

# Set up axes
ax.set_xticklabels([''] + all_categories, rotation=90)
ax.set_yticklabels([''] + all_categories)

# Force label at every tick
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# sphinx_gallery_thumbnail_number = 2
plt.show()
```

```
/tmp/ipykernel_22615/3585656379.py:33: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_xticklabels([''] + all_categories, rotation=90)
/tmp/ipykernel_22615/3585656379.py:34: UserWarning: FixedFormatter should only
be used together with FixedLocator
  ax.set_yticklabels([''] + all_categories)
```




[46]:
```python
def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

        # Get top N categories
        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []

        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('(%.2f) %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])

predict('rain')
predict('raining')
predict('rained')
predict('rains')
```

```
> rain
(-1.47) JJ
(-1.80) NNP
```

```
(-2.44) UH

> raining
(-0.04) VBG
(-4.67) JJ
(-4.73) NNP

> rained
(-1.49) NNP
(-1.81) JJ
(-2.20) VBN

> rains
(-0.53) NNPS
(-1.38) NNS
(-3.63) PRP$
```

[ ]:

```

```