

Introduction to PyTorch

Jianan Zhao

Mila - Quebec AI Institute

Email: jianan.zhao@mila.quebec

Contributors: Jiarui Lu, Meng Qu, Zhaocheng Zhu, Louis-Pascal Xhonneux, Shengchao Liu and Andreea Deac



WHAT DO WE WANT?



**BUILD
NEURAL NETWORKS**



WHAT DON'T WE WANT?



MATH



HOW CAN WE DO THAT?



USE DL FRAMEWORKS



Deep learning ecosystem

- CPU
 - Any language is applicable, Python, Matlab, R, C++, Java, etc.
 - Prefer languages with support to matrix(tensor) operation like Python, Matlab, R
 - Python has some nice packages/libraries
 - NumPy: matrix operation
 - Scikit-Learn: classic machine learning models
- GPU
 - Python has a large number of options for deep learning frameworks/packages: pytorch/tensorflow/keras/mxnet ...

Researchers keep contributing to this ecosystem: adding deep learning packages, adding models, etc.

How powerful are deep learning frameworks?

2012

More than 3000 lines

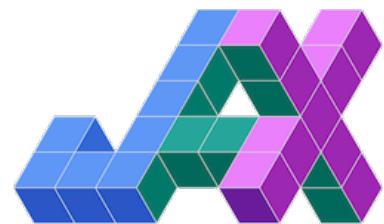


```
1429 }
1430 if (stepbptt+bptt_block>=3)
1431 for (a=0; a<layer1_size; a++) {
1432     neu[a].acbptt_hidden
1433     [(step+1)*layer1_size+a].ac;
1434
1435     neu# [a+layer0_size-layer1_size]
1436     .acbptt_hidden
1437     [(step+2)*layer1_size+a].ac;
1438 }
1439
1440 for (a=0;
1441     ac(bptt+bptt_block)*layer1_size;
1442     a++) {
1443     bptt_hidden[a].er=0;
1444
1445     for (b=0; b<layer1_size; b++)
1446         neu[b].ac=bptt_hidden[b].ac;
1447         //restore hidden layer after
1448         bptt
1449
1450     //for (b=0; b<layer1_size; b++)
1451     //copy temporary syn#
1452     if ((counter%10)==0) {
1453         for (a=layer0_size-layer1_size;
1454             a<layer0_size; a++) {
1455             syn# [a+b*layer0_size]
1456             .ac=bptt_hidden[a].syn#
1457             [a+layer0_size];
1458             .weight
1459             syn# [a+b*layer0_size]
1460             .weight*beta2;
1461
1462             bptt_hidden[a].syn#
1463             .weight
1464             syn# [a+b*layer0_size]
1465             .weight*beta2;
1466
1467             bptt_hidden[a].syn#
1468             .weight
1469             syn# [a+b*layer0_size]
1470             .weight*beta2;
1471
1472             bptt_hidden[a].syn#
1473             .weight
1474             syn# [a+b*layer0_size]
1475             .weight*beta2;
1476
1477             bptt_hidden[a].syn#
1478             .weight
1479             syn# [a+b*layer0_size]
1480             .weight*beta2;
1481
1482             bptt_hidden[a].syn#
1483             .weight
1484             syn# [a+b*layer0_size]
1485             .weight*beta2;
1486
1487             bptt_hidden[a].syn#
1488             .weight
1489             syn# [a+b*layer0_size]
1490             .weight*beta2;
1491
1492             bptt_hidden[a].syn#
1493             .weight
1494             syn# [a+b*layer0_size]
1495             .weight*beta2;
1496
1497             bptt_hidden[a].syn#
1498             .weight
1499             syn# [a+b*layer0_size]
1500             .weight*beta2;
1501
1502             bptt_hidden[a].syn#
1503             .weight
1504             syn# [a+b*layer0_size]
1505             .weight*beta2;
1506
1507             bptt_hidden[a].syn#
1508             .weight
1509             syn# [a+b*layer0_size]
1510             .weight*beta2;
1511
1512             bptt_hidden[a].syn#
1513             .weight
1514             syn# [a+b*layer0_size]
1515             .weight*beta2;
1516
1517             bptt_hidden[a].syn#
1518             .weight
1519             syn# [a+b*layer0_size]
1520             .weight*beta2;
1521
1522             bptt_hidden[a].syn#
1523             .weight
1524             syn# [a+b*layer0_size]
1525             .weight*beta2;
1526
1527             bptt_hidden[a].syn#
1528             .weight
1529             syn# [a+b*layer0_size]
1530             .weight*beta2;
1531
1532             start=clock();
1533             while (1) {
1534                 counter++;
1535
1536                 if ((counter%10000)==0) if
1537                     (debug_mode>1) {
1538                     now=clock();
1539                     if (train_words>0)
1540                         printf("%d\tAlpha: %f\t"
1541                             "TRAIN entropy: %f\t"
1542                             "Progress: %d%%\n",
1543                             Words/sec: %1.1f ", 13,
1544                             iter, alpha,
1545                             -log(log10(2)/counter,
1546                                 counter),
1547                             (real)train_words*100,
1548                             counter/((double)
1549                             (now-start)/100000.0));
1550
1551                 else {
1552                     printf("%d\tAlpha: %f\t"
1553                         "TRAIN entropy: %f\t"
1554                         "Progress: %d%\n",
1555                         Words/sec: %1.1f ", 13,
1556                         iter, alpha,
1557                         -log(log10(2)/counter,
1558                             counter),
1559                         (real)train_words*100,
1560                         counter/((double)
1561                         (now-start)/100000.0));
1562
1563                 wordcn++;
1564
1565                 if (prob_other!=0) {
1566                     log_combine+=log10(neu2
1567                         [word].ac*lambda +
1568                         prob_other*(1-lambda));
1569
1570                     senp+=log10(neu2[word].ac*lambda +
1571                         prob_other*(1-lambda));
1572
1573                     wordcn++;
1574
1575                 else {
1576                     //assign to OOVs some score to
1577                     //correctly rescore nbest lists,
1578                     //reasonable value can be less
1579                     //than 1/V or backoff LM score
1580                     //in case it is trained on more
1581                     //data)
1582                     //this means that PPL results from
1583                     //nbest list rescoring are not
1584                     //true probabilities anymore (as
1585                     //in open vocabulary LMs)
1586
1587                     real oov_penalty=-5; //log
1588                     penalty
1589
1590                     if (prob_other!=0) {
1591                         logp+=log10(prob_other);
1592                         log_other+=log10(prob_other);
1593                         log_combine+=log10(prob_other);
1594                         senp+=log10(prob_other);
1595
1596                     } else {
1597                         logp+=oov_penalty;
1598                         log_other+=oov_penalty;
1599                         log_combine+=oov_penalty;
1600                         senp+=oov_penalty;
1601
1602                     wordcn++;
1603
1604                     if (prob_other!=0) {
1605                         logp+=log10(prob_other);
1606                         log_other+=log10(prob_other);
1607                         log_combine+=log10(prob_other);
1608                         senp+=log10(prob_other);
1609
1610                     } else {
1611                         logp+=oov_penalty;
1612                         log_other+=oov_penalty;
1613                         log_combine+=oov_penalty;
1614                         senp+=oov_penalty;
1615
1616                     wordcn++;
1617
1618                     if (prob_other!=0) {
1619                         logp+=log10(prob_other);
1620                         log_other+=log10(prob_other);
1621                         log_combine+=log10(prob_other);
1622                         senp+=log10(prob_other);
1623
1624                     } else {
1625                         logp+=oov_penalty;
1626                         log_other+=oov_penalty;
1627                         log_combine+=oov_penalty;
1628                         senp+=oov_penalty;
1629
1630                     wordcn++;
1631
1632                     if (prob_other!=0) {
1633                         logp+=log10(prob_other);
1634                         log_other+=log10(prob_other);
1635                         log_combine+=log10(prob_other);
1636                         senp+=log10(prob_other);
1637
1638                     } else {
1639                         logp+=oov_penalty;
1640                         log_other+=oov_penalty;
1641                         log_combine+=oov_penalty;
1642                         senp+=oov_penalty;
1643
1644                     wordcn++;
1645
1646                     if (prob_other!=0) {
1647                         logp+=log10(prob_other);
1648                         log_other+=log10(prob_other);
1649                         log_combine+=log10(prob_other);
1650                         senp+=log10(prob_other);
1651
1652                     } else {
1653                         logp+=oov_penalty;
1654                         log_other+=oov_penalty;
1655                         log_combine+=oov_penalty;
1656                         senp+=oov_penalty;
1657
1658                     wordcn++;
1659
1660                     if (prob_other!=0) {
1661                         logp+=log10(prob_other);
1662                         log_other+=log10(prob_other);
1663                         log_combine+=log10(prob_other);
1664                         senp+=log10(prob_other);
1665
1666                     } else {
1667                         logp+=oov_penalty;
1668                         log_other+=oov_penalty;
1669                         log_combine+=oov_penalty;
1670                         senp+=oov_penalty;
1671
1672                     wordcn++;
1673
1674                     if (prob_other!=0) {
1675                         logp+=log10(prob_other);
1676                         log_other+=log10(prob_other);
1677                         log_combine+=log10(prob_other);
1678                         senp+=log10(prob_other);
1679
1680                     } else {
1681                         logp+=oov_penalty;
1682                         log_other+=oov_penalty;
1683                         log_combine+=oov_penalty;
1684                         senp+=oov_penalty;
1685
1686                     wordcn++;
1687
1688                     if (prob_other!=0) {
1689                         logp+=log10(prob_other);
1690                         log_other+=log10(prob_other);
1691                         log_combine+=log10(prob_other);
1692                         senp+=log10(prob_other);
1693
1694                     } else {
1695                         logp+=oov_penalty;
1696                         log_other+=oov_penalty;
1697                         log_combine+=oov_penalty;
1698                         senp+=oov_penalty;
1699
1700                     wordcn++;
1701
1702                     if (prob_other!=0) {
1703                         logp+=log10(prob_other);
1704                         log_other+=log10(prob_other);
1705                         log_combine+=log10(prob_other);
1706                         senp+=log10(prob_other);
1707
1708                     } else {
1709                         logp+=oov_penalty;
1710                         log_other+=oov_penalty;
1711                         log_combine+=oov_penalty;
1712                         senp+=oov_penalty;
1713
1714                     wordcn++;
1715
1716                     if (prob_other!=0) {
1717                         logp+=log10(prob_other);
1718                         log_other+=log10(prob_other);
1719                         log_combine+=log10(prob_other);
1720                         senp+=log10(prob_other);
1721
1722                     } else {
1723                         logp+=oov_penalty;
1724                         log_other+=oov_penalty;
1725                         log_combine+=oov_penalty;
1726                         senp+=oov_penalty;
1727
1728                     wordcn++;
1729
1730                     if (prob_other!=0) {
1731                         logp+=log10(prob_other);
1732                         log_other+=log10(prob_other);
1733                         log_combine+=log10(prob_other);
1734                         senp+=log10(prob_other);
1735
1736                     } else {
1737                         logp+=oov_penalty;
1738                         log_other+=oov_penalty;
1739                         log_combine+=oov_penalty;
1740                         senp+=oov_penalty;
1741
1742                     wordcn++;
1743
1744                     if (prob_other!=0) {
1745                         logp+=log10(prob_other);
1746                         log_other+=log10(prob_other);
1747                         log_combine+=log10(prob_other);
1748                         senp+=log10(prob_other);
1749
1750                     } else {
1751                         logp+=oov_penalty;
1752                         log_other+=oov_penalty;
1753                         log_combine+=oov_penalty;
1754                         senp+=oov_penalty;
1755
1756                     wordcn++;
1757
1758                     if (prob_other!=0) {
1759                         logp+=log10(prob_other);
1760                         log_other+=log10(prob_other);
1761                         log_combine+=log10(prob_other);
1762                         senp+=log10(prob_other);
1763
1764                     } else {
1765                         logp+=oov_penalty;
1766                         log_other+=oov_penalty;
1767                         log_combine+=oov_penalty;
1768                         senp+=oov_penalty;
1769
1770                     wordcn++;
1771
1772                     if (prob_other!=0) {
1773                         logp+=log10(prob_other);
1774                         log_other+=log10(prob_other);
1775                         log_combine+=log10(prob_other);
1776                         senp+=log10(prob_other);
1777
1778                     } else {
1779                         logp+=oov_penalty;
1780                         log_other+=oov_penalty;
1781                         log_combine+=oov_penalty;
1782                         senp+=oov_penalty;
1783
1784                     wordcn++;
1785
1786                     if (prob_other!=0) {
1787                         logp+=log10(prob_other);
1788                         log_other+=log10(prob_other);
1789                         log_combine+=log10(prob_other);
1790                         senp+=log10(prob_other);
1791
1792                     } else {
1793                         logp+=oov_penalty;
1794                         log_other+=oov_penalty;
1795                         log_combine+=oov_penalty;
1796                         senp+=oov_penalty;
1797
1798                     wordcn++;
1799
1800                     if (prob_other!=0) {
1801                         logp+=log10(prob_other);
1802                         log_other+=log10(prob_other);
1803                         log_combine+=log10(prob_other);
1804                         senp+=log10(prob_other);
1805
1806                     } else {
1807                         logp+=oov_penalty;
1808                         log_other+=oov_penalty;
1809                         log_combine+=oov_penalty;
1810                         senp+=oov_penalty;
1811
1812                     wordcn++;
1813
1814                     if (prob_other!=0) {
1815                         logp+=log10(prob_other);
1816                         log_other+=log10(prob_other);
1817                         log_combine+=log10(prob_other);
1818                         senp+=log10(prob_other);
1819
1820                     } else {
1821                         logp+=oov_penalty;
1822                         log_other+=oov_penalty;
1823                         log_combine+=oov_penalty;
1824                         senp+=oov_penalty;
1825
1826                     wordcn++;
1827
1828                     if (prob_other!=0) {
1829                         logp+=log10(prob_other);
1830                         log_other+=log10(prob_other);
1831                         log_combine+=log10(prob_other);
1832                         senp+=log10(prob_other);
1833
1834                     } else {
1835                         logp+=oov_penalty;
1836                         log_other+=oov_penalty;
1837                         log_combine+=oov_penalty;
1838                         senp+=oov_penalty;
1839
1840                     wordcn++;
1841
1842                     if (prob_other!=0) {
1843                         logp+=log10(prob_other);
1844                         log_other+=log10(prob_other);
1845                         log_combine+=log10(prob_other);
1846                         senp+=log10(prob_other);
1847
1848                     } else {
1849                         logp+=oov_penalty;
1850                         log_other+=oov_penalty;
1851                         log_combine+=oov_penalty;
1852                         senp+=oov_penalty;
1853
1854                     wordcn++;
1855
1856                     if (prob_other!=0) {
1857                         logp+=log10(prob_other);
1858                         log_other+=log10(prob_other);
1859                         log_combine+=log10(prob_other);
1860                         senp+=log10(prob_other);
1861
1862                     } else {
1863                         logp+=oov_penalty;
1864                         log_other+=oov_penalty;
1865                         log_combine+=oov_penalty;
1866                         senp+=oov_penalty;
1867
1868                     wordcn++;
1869
1870                     if (prob_other!=0) {
1871                         logp+=log10(prob_other);
1872                         log_other+=log10(prob_other);
1873                         log_combine+=log10(prob_other);
1874                         senp+=log10(prob_other);
1875
1876                     } else {
1877                         logp+=oov_penalty;
1878                         log_other+=oov_penalty;
1879                         log_combine+=oov_penalty;
1880                         senp+=oov_penalty;
1881
1882                     wordcn++;
1883
1884                     if (prob_other!=0) {
1885                         logp+=log10(prob_other);
1886                         log_other+=log10(prob_other);
1887                         log_combine+=log10(prob_other);
1888                         senp+=log10(prob_other);
1889
1890                     } else {
1891                         logp+=oov_penalty;
1892                         log_other+=oov_penalty;
1893                         log_combine+=oov_penalty;
1894                         senp+=oov_penalty;
1895
1896                     wordcn++;
1897
1898                     if (prob_other!=0) {
1899                         logp+=log10(prob_other);
1900                         log_other+=log10(prob_other);
1901                         log_combine+=log10(prob_other);
1902                         senp+=log10(prob_other);
1903
1904                     } else {
1905                         logp+=oov_penalty;
1906                         log_other+=oov_penalty;
1907                         log_combine+=oov_penalty;
1908                         senp+=oov_penalty;
1909
1910                     wordcn++;
1911
1912                     if (prob_other!=0) {
1913                         logp+=log10(prob_other);
1914                         log_other+=log10(prob_other);
1915                         log_combine+=log10(prob_other);
1916                         senp+=log10(prob_other);
1917
1918                     } else {
1919                         logp+=oov_penalty;
1920                         log_other+=oov_penalty;
1921                         log_combine+=oov_penalty;
1922                         senp+=oov_penalty;
1923
1924                     wordcn++;
1925
1926                     if (prob_other!=0) {
1927                         logp+=log10(prob_other);
1928                         log_other+=log10(prob_other);
1929                         log_combine+=log10(prob_other);
1930                         senp+=log10(prob_other);
1931
1932                     } else {
1933                         logp+=oov_penalty;
1934                         log_other+=oov_penalty;
1935                         log_combine+=oov_penalty;
1936                         senp+=oov_penalty;
1937
1938                     wordcn++;
1939
1940                     if (prob_other!=0) {
1941                         logp+=log10(prob_other);
1942                         log_other+=log10(prob_other);
1943                         log_combine+=log10(prob_other);
1944                         senp+=log10(prob_other);
1945
1946                     } else {
1947                         logp+=oov_penalty;
1948                         log_other+=oov_penalty;
1949                         log_combine+=oov_penalty;
1950                         senp+=oov_penalty;
1951
1952                     wordcn++;
1953
1954                     if (prob_other!=0) {
1955                         logp+=log10(prob_other);
1956                         log_other+=log10(prob_other);
1957                         log_combine+=log10(prob_other);
1958                         senp+=log10(prob_other);
1959
1960                     } else {
1961                         logp+=oov_penalty;
1962                         log_other+=oov_penalty;
1963                         log_combine+=oov_penalty;
1964                         senp+=oov_penalty;
1965
1966                     wordcn++;
1967
1968                     if (prob_other!=0) {
1969                         logp+=log10(prob_other);
1970                         log_other+=log10(prob_other);
1971                         log_combine+=log10(prob_other);
1972                         senp+=log10(prob_other);
1973
1974                     } else {
1975                         logp+=oov_penalty;
1976                         log_other+=oov_penalty;
1977                         log_combine+=oov_penalty;
1978                         senp+=oov_penalty;
1979
1980                     wordcn++;
1981
1982                     if (prob_other!=0) {
1983                         logp+=log10(prob_other);
1984                         log_other+=log10(prob_other);
1985                         log_combine+=log10(prob_other);
1986                         senp+=log10(prob_other);
1987
1988                     } else {
1989                         logp+=oov_penalty;
1990                         log_other+=oov_penalty;
1991                         log_combine+=oov_penalty;
1992                         senp+=oov_penalty;
1993
1994                     wordcn++;
1995
1996                     if (prob_other!=0) {
1997                         logp+=log10(prob_other);
1998                         log_other+=log10(prob_other);
1999                         log_combine+=log10(prob_other);
2000                         senp+=log10(prob_other);
2001
2002                     } else {
2003                         logp+=oov_penalty;
2004                         log_other+=oov_penalty;
2005                         log_combine+=oov_penalty;
2006                         senp+=oov_penalty;
2007
2008                     wordcn++;
2009
2010                     if (prob_other!=0) {
2011                         logp+=log10(prob_other);
2012                         log_other+=log10(prob_other);
2013                         log_combine+=log10(prob_other);
2014                         senp+=log10(prob_other);
2015
2016                     } else {
2017                         logp+=oov_penalty;
2018                         log_other+=oov_penalty;
2019                         log_combine+=oov_penalty;
2020                         senp+=oov_penalty;
2021
2022                     wordcn++;
2023
2024                     if (prob_other!=0) {
2025                         logp+=log10(prob_other);
2026                         log_other+=log10(prob_other);
2027                         log_combine+=log10(prob_other);
2028                         senp+=log10(prob_other);
2029
2030                     } else {
2031                         logp+=oov_penalty;
2032                         log_other+=oov_penalty;
2033                         log_combine+=oov_penalty;
2034                         senp+=oov_penalty;
2035
2036                     wordcn++;
2037
2038                     if (prob_other!=0) {
2039                         logp+=log10(prob_other);
2040                         log_other+=log10(prob_other);
2041                         log_combine+=log10(prob_other);
2042                         senp+=log10(prob_other);
2043
2044                     } else {
2045                         logp+=oov_penalty;
2046                         log_other+=oov_penalty;
2047                         log_combine+=oov_penalty;
2048                         senp+=oov_penalty;
2049
2050                     wordcn++;
2051
2052                     if (prob_other!=0) {
2053                         logp+=log10(prob_other);
2054                         log_other+=log10(prob_other);
2055                         log_combine+=log10(prob_other);
2056                         senp+=log10(prob_other);
2057
2058                     } else {
2059                         logp+=oov_penalty;
2060                         log_other+=oov_penalty;
2061                         log_combine+=oov_penalty;
2062                         senp+=oov_penalty;
2063
2064                     wordcn++;
2065
2066                     if (prob_other!=0) {
2067                         logp+=log10(prob_other);
2068                         log_other+=log10(prob_other);
2069                         log_combine+=log10(prob_other);
2070                         senp+=log10(prob_other);
2071
2072                     } else {
2073                         logp+=oov_penalty;
2074                         log_other+=oov_penalty;
2075                         log_combine+=oov_penalty;
2076                         senp+=oov_penalty;
2077
2078                     wordcn++;
2079
2080                     if (prob_other!=0) {
2081                         logp+=log10(prob_other);
2082                         log_other+=log10(prob_other);
2083                         log_combine+=log10(prob_other);
2084                         senp+=log10(prob_other);
2085
2086                     } else {
2087                         logp+=oov_penalty;
2088                         log_other+=oov_penalty;
2089                         log_combine+=oov_penalty;
2090                         senp+=oov_penalty;
2091
2092                     wordcn++;
2093
2094                     if (prob_other!=0) {
2095                         logp+=log10(prob_other);
2096                         log_other+=log10(prob_other);
2097                         log_combine+=log10(prob_other);
2098                         senp+=log10(prob_other);
2099
2100                     } else {
2101                         logp+=oov_penalty;
2102                         log_other+=oov_penalty;
2103                         log_combine+=oov_penalty;
2104                         senp+=oov_penalty;
2105
2106                     wordcn++;
2107
2108                     if (prob_other!=0) {
2109                         logp+=log10(prob_other);
2110                         log_other+=log10(prob_other);
2111                         log_combine+=log10(prob_other);
2112                         senp+=log10(prob_other);
2113
2114                     } else {
2115                         logp+=oov_penalty;
2116                         log_other+=oov_penalty;
2117                         log_combine+=oov_penalty;
2118                         senp+=oov_penalty;
2119
2120                     wordcn++;
2121
2122                     if (prob_other!=0) {
2123                         logp+=log10(prob_other);
2124                         log_other+=log10(prob_other);
2125                         log_combine+=log10(prob_other);
2126                         senp+=log10(prob_other);
2127
2128                     } else {
2129                         logp+=oov_penalty;
2130                         log_other+=oov_penalty;
2131                         log_combine+=oov_penalty;
2132                         senp+=oov_penalty;
2133
2134                     wordcn++;
2135
2136                     if (prob_other!=0) {
2137                         logp+=log10(prob_other);
2138                         log_other+=log10(prob_other);
2139                         log_combine+=log10(prob_other);
2140                         senp+=log10(prob_other);
2141
2142                     } else {
2143                         logp+=oov_penalty;
2144                         log_other+=oov_penalty;
2145                         log_combine+=oov_penalty;
2146                         senp+=oov_penalty;
2147
2148                     wordcn++;
2149
2150                     if (prob_other!=0) {
2151                         logp+=log10(prob_other);
2152                         log_other+=log10(prob_other);
2153                         log_combine+=log10(prob_other);
2154                         senp+=log10(prob_other);
2155
2156                     } else {
2157                         logp+=oov_penalty;
2158                         log_other+=oov_penalty;
2159                         log_combine+=oov_penalty;
2160                         senp+=oov_penalty;
2161
2162                     wordcn++;
2163
2164                     if (prob_other!=0) {
2165                         logp+=log10(prob_other);
2166                         log_other+=log10(prob_other);
2167                         log_combine+=log10(prob_other);
2168                         senp+=log10(prob_other);
2169
2170                     } else {
2171                         logp+=oov_penalty;
2172                         log_other+=oov_penalty;
2173                         log_combine+=oov_penalty;
2174                         senp+=oov_penalty;
2175
2176                     wordcn++;
2177
2178                     if (prob_other!=0) {
2179                         logp+=log10(prob_other);
2180                         log_other+=log10(prob_other);
2181                         log_combine+=log10(prob_other);
2182                         senp+=log10(prob_other);
2183
2184                     } else {
2185                         logp+=oov_penalty;
2186                         log_other+=oov_penalty;
2187                         log_combine+=oov_penalty;
2188                         senp+=oov_penalty;
2189
2190                     wordcn++;
2191
2192                     if (prob_other!=0) {
2193                         logp+=log10(prob_other);
2194                         log_other+=log10(prob_other);
2195                         log_combine+=log10(prob_other);
2196                         senp+=log10(prob_other);
2197
2198                     } else {
2199                         logp+=oov_penalty;
2200                         log_other+=oov_penalty;
2201                         log_combine+=oov_penalty;
2202                         senp+=oov_penalty;
2203
2204                     wordcn++;
2205
2206                     if (prob_other!=0) {
2207                         logp+=log10(prob_other);
2208                         log_other+=log10(prob_other);
2209                         log_combine+=log10(prob_other);
2210                         senp+=log10(prob_other);
2211
2212                     } else {
2213                         logp+=oov_penalty;
2214                         log_other+=oov_penalty;
2215                         log_combine+=oov_penalty;
2216                         senp+=oov_penalty;
2217
2218                     wordcn++;
2219
2220                     if (prob_other!=0) {
2221                         logp+=log10(prob_other);
2222                         log_other+=log10(prob_other);
2223                         log_combine+=log10(prob_other);
2224                         senp+=log10(prob_other);
2225
2226                     } else {
2227                         logp+=oov_penalty;
2228                         log_other+=oov_penalty;
2229                         log_combine+=oov_penalty;
2230                         senp+=oov_penalty;
2231
2232                     wordcn++;
2233
2234                     if (prob_other!=0) {
2235                         logp+=log10(prob_other);
2236                         log_other+=log10(prob_other);
2237                         log_combine+=log10(prob_other);
2238                         senp+=log10(prob_other);
2239
2240                     } else {
2241                         logp+=oov_penalty;
2242                         log_other+=oov_penalty;
2243                         log_combine+=oov_penalty;
2244                         senp+=oov_penalty;
2245
2246                     wordcn++;
2247
2248                     if (prob_other!=0) {
2249                         logp+=log10(prob_other);
2250                         log_other+=log10(prob_other);
2251                         log_combine+=log10(prob_other);
2252                         senp+=log10(prob_other);
2253
2254                     } else {
2255                         logp+=oov_penalty;
2256                         log_other+=oov_penalty;
2257                         log_combine+=oov_penalty;
2258                         senp+=oov_penalty;
2259
2260                     wordcn++;
2261
2262                     if (prob_other!=0) {
2263                         logp+=log10(prob_other);
2264                         log_other+=log10(prob_other);
2265                         log_combine+=log10(prob_other);
2266                         senp+=log10(prob_other);
2267
2268                     } else {
2269                         logp+=oov_penalty;
2270                         log_other+=oov_penalty;
2271                         log_combine+=oov_penalty;
2272                         senp+=oov_penalty;
2273
2274                     wordcn++;
2275
2276                     if (prob_other!=0) {
2277                         logp+=log10(prob_other);
2278                         log_other+=log10(prob_other);
2279                         log_combine+=log10(prob_other);
2280                         senp+=log10(prob_other);
2281
2282                     } else {
2283                         logp+=oov_penalty;
2284                         log_other+=oov_penalty;
2285                         log_combine+=oov_penalty;
2286                         senp+=oov_penalty;
2287
2288                     wordcn++;
2289
2290                     if (prob_other!=0) {
2291                         logp+=log10(prob_other);
2292                         log_other+=log10(prob_other);
2293                         log_combine+=log10(prob_other);
2294                         senp+=log10(prob_other);
2295
2296                     } else {
2297                         logp+=oov_penalty;
2298                         log_other+=oov_penalty;
2299                         log_combine+=oov_penalty;
2300                         senp+=oov_penalty;
2301
2302                     wordcn++;
2303
2304                     if (prob_other!=0) {
2305                         logp+=log10(prob_other);
2306                         log_other+=log10(prob_other);
2307                         log_combine+=log10(prob_other);
2308                         senp+=log10(prob_other);
2309
2310                     } else {
2311                         logp+=oov_penalty;
2312                         log_other+=oov_penalty;
2313                         log_combine+=oov_penalty;
2314                         senp+=oov_penalty;
2315
2316                     wordcn++;
2317
2318                     if (prob_other!=0) {
2319                         logp+=log10(prob_other);
2320                         log_other+=log10(prob_other);
2321                         log_combine+=log10(prob_other);
2322                         senp+=log10(prob_other);
2323
2324                     } else {
2325                         logp+=oov_penalty;
2326                         log_other+=oov_penalty;
2327                         log_combine+=oov_penalty;
2328                         senp+=oov_penalty;
2329
2330                     wordcn++;
2331
2332                     if (prob_other!=0) {
2333                         logp+=log10(prob_other);
2334                         log_other+=log10(prob_other);
2335                         log_combine+=log10(prob_other);
2336                         senp+=log10(prob_other);
2337
2338                     } else {
2339                         logp+=oov_penalty;
2340                         log_other+=oov_penalty;
2341                         log_combine+=oov_penalty;
2342                         senp+=oov_penalty;
2343
2344                     wordcn++;
2345
2346                     if (prob_other!=0) {
2347                         logp+=log10(prob_other);
2348                         log_other+=log10(prob_other);
2349                         log_combine+=log10(prob_other);
2350                         senp+=log10(prob_other);
2351
2352                     } else {
2353                         logp+=oov_penalty;
2354                         log_other+=oov_penalty;
2355                         log_combine+=oov_penalty;
2356                         senp+=oov_penalty;
2357
2358                     wordcn++;
2359
2360                     if (prob_other!=0) {
2361                         logp+=log10(prob_other);
2362                         log_other+=log10(prob_other);
2363                         log_combine+=log10(prob_other);
2364                         senp+=log10(prob_other);
2365
2366                     } else {
2367                         logp+=oov_penalty;
2368                         log_other+=oov_penalty;
2369                         log_combine+=oov_penalty;
2370                         senp+=oov_penalty;
2371
2372                     wordcn++;
2373
2374                     if (prob_other!=0) {
2375                         logp+=log10(prob_other);
2376                         log_other+=log10(prob_other);
2377                         log_combine+=log10(prob_other);
2378                         senp+=log10(prob_other);
2379
2380                     } else {
2381                         logp+=oov_penalty;
2382                         log_other+=oov_penalty;
2383                         log_combine+=oov_penalty;
2384                         senp+=oov_penalty;
2385
2386                     wordcn++;
2387
2388                     if (prob_other!=0) {
2389                         logp+=log10(prob_other);
2390                         log_other+=log10(prob_other);
2391                         log_combine+=log10(prob_other);
2392                         senp+=log10(prob_other);
2393
2394                     } else {
2395                         logp+=oov_penalty;
2396                         log_other+=oov_penalty;
2397                         log_combine+=oov_penalty;
2398                         senp+=oov_penalty;
2399
2400                     wordcn++;
2401
2402                     if (prob_other!=0) {
2403                         logp+=log10(prob_other);
2404                         log_other+=log10(prob_other);
2405                         log_combine+=log10(prob_other);
2406                         senp+=log10(prob_other);
2407
2408                     } else {
2409                         logp+=oov_penalty;
2410                         log_other+=oov_penalty;
2411                         log_combine+=oov_penalty;
2412                         senp+=oov_penalty;
2413
2414                     wordcn++;
2415
2416                     if (prob_other!=0) {
2417                         logp+=log10(prob_other);
2418                         log_other+=log10(prob_other);
2419                         log_combine+=log10(prob_other);
2420                         senp+=log10(prob_other);
2421
2422                     } else {
2423                         logp+=oov_penalty;
2424                         log_other+=oov_penalty;
2425                         log_combine+=oov_penalty;
2426                         senp+=oov_penalty;
2427
2428                     wordcn++;
2429
2430                     if (prob_other!=0) {
2431                         logp+=log10(prob_other);
2432                         log_other+=log10(prob_other);
2433                         log_combine+=log10(prob_other);
2434                         senp+=log10(prob_other);
2435
2436                     } else {
2437                         logp+=oov_penalty;
2438                         log_other+=oov_penalty;
2439                         log_combine+=oov_penalty;
2440                         senp+=oov_penalty;
2441
2442                     wordcn++;
2443
2444                     if (prob_other!=0) {
2445                         logp+=log
```

Why deep learning frameworks?

- DL Frameworks can help us
 - build neural networks without annoying math
 - reduce development efforts on standard modules
 - accelerate training with GPUs or distributed training
- e.g. You can apply standard models to your own dataset with in ~10 lines of Python code
- Also most open source projects are based on these frameworks

Popular deep learning frameworks



History on deep learning frameworks

- Theano, 2007 - 2017, by UdeM (Yoshua Begin, Ian Godfellow, etc.)
- Caffe, 2013, by Yangqing Jia
 - Caffe2, 2017, by Facebook / Yangqing Jia
- Keras, 2015, by Francois Chollet, first built on Theano as engine (now adapts to TensorFlow)
- TensorFlow, 2015, by Google
- MXNet, 2015, by Mu Li (CMU) /Amazon
- CNTK, 2016, by Microsoft
- **PyTorch, 2017, by Facebook**
- Jax, 2019, by Google/Deepmind

Which framework to use?



- high performance
- good distributed & large-scale training
- great for industrial deployment



- difficult to get started with
- difficult to debug



- similar to native Python logic
 - easy to get start with
 - easy to debug
-
- bad support of large-scale training
 - write bad performance code unintentionally

In this class

- We will focus on **PyTorch**
 - PyTorch is “pythonic” in its style
 - PyTorch is opensource backed by Facebook
- It provides us with Tensors, Autodifferentiation, and functions commonly used in Deep Learning models.

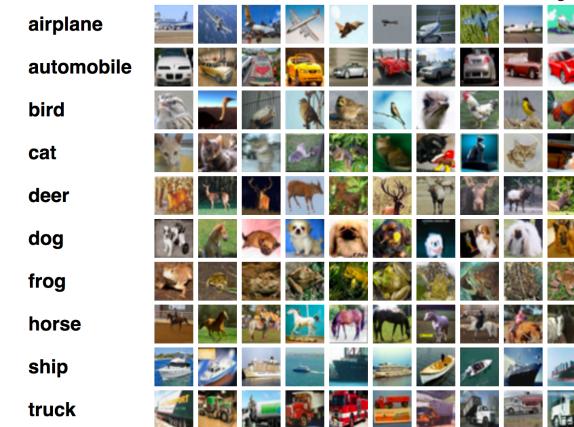
The goal of (supervised) deep learning

- Transform data from one representation to another

- Example 1: Image Classification

- Input: images
- Output: image labels

- Example 2: Sentence Regression
 - Input: item reviews
 - Output: corresponding ratings



★★★★★ The GBC (Goodfellow, Bengio and Courville) book is worth the reading!

Reviewed in Canada on February 25, 2017

Format: Hardcover | **Verified Purchase**

It's definitely THE authoritative reference on Deep Learning but you should not be allergic to maths. That said reinforcement learning is superficially exposed which is due for an additional chapter [Note].

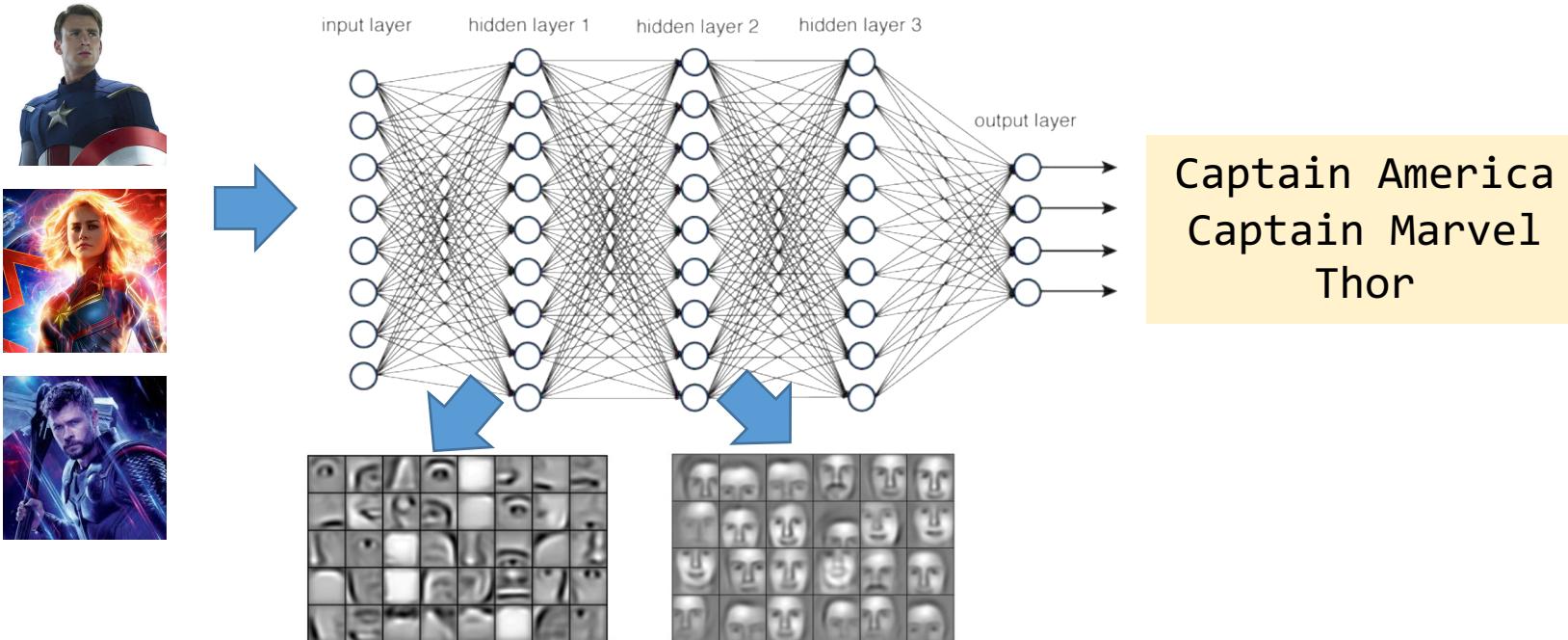
The main weakness of this masterpiece is the lack of practical programming exercises left to a companion web site. But to cover all the practical stuff, the book should have exceeded 775 pages that it already has.

I dream of the same content in the form of a series of iPython Notebooks with all exercises and code samples using Keras, TensorFlow and Theano.

[Note] To be completely honest the authors wrote a short disclaimer in the «Machine Learning Basics» chapter 5, page 103 about reinforcement learning. « Such algorithms are beyond the scope of this book ».

Workflow of deep learning

- We want to train a neural network f_θ for a given task T



- We solve $\min_{\theta} L(f_\theta(x), y)$ using gradient descent $\theta_{i+1} = \theta_i - \epsilon \nabla_{\theta} L(f_\theta(x), y)$

Workflow of deep learning

- What we need/want:
 - A way to hold the data (x, y)
 - Functions to code the neural network $f_\theta(x)$
 - Functions to compute the loss $L(f_\theta(x), y)$
 - The ability to compute $\nabla_\theta L(f_\theta(x), y)$ automatically without needing to do maths on paper apriori
- PyTorch gives us these things through
 - `Torch.tensor` and `torch.utils.data.DataLoader` (to load data from files)
 - `Torch.nn`
 - `Torch.nn.functional` (or `Torch.nn`)
 - `Tensor.backward()` (see `torch.autograd.backward`)



Workflow of deep learning

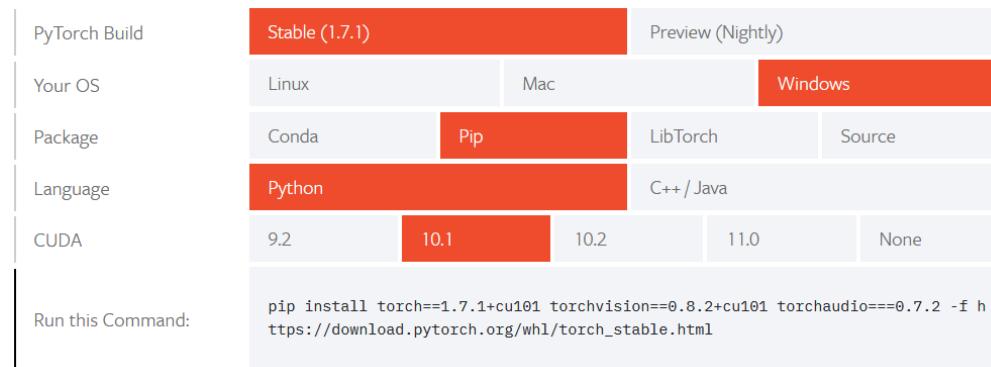
- Installation
 - Libraries for Python
- Data preparation
 - Know how PyTorch stores data / what the data look like
 - Split the data into train/valid/test
- Model preparation
- Model training
 - Define loss function
 - Choose optimization method
 - Train the model on training data
- Model evaluation
 - Evaluate the model on test data

Ready, steady, go

- We will learn how to use PyTorch to build and train Neural Networks
- Tutorial code for the first half can be found at:
 - <https://bit.ly/3Hk4ihL> (Notebook Part I)

Installation

- On your laptop go to <https://pytorch.org/get-started/locally/>



- Use the stable build, your OS, either Pip or Conda, and the cuda version you have if you have a Nvidia GPU in your laptop
- **On Google Colab**
 - Faster training with a GPU!
 - Enable it in **Edit -> Notebook settings -> Hardware accelerator**

Installation

- Let's check if they are installed properly.

```
>>> import torch  
>>> import torchvision
```

- If nothing complains, then you are ready to go.
- To check if GPU acceleration is available,

```
>>> torch.cuda.is_available()
```
- Note this doesn't necessarily mean everything runs on GPU by default!

PyTorch basics

- Idea of deep learning <https://pytorch.org/docs/stable/index.html>
- A fundamental data structure: Tensor
- Training through auto-gradient
- An example of linear regression

Data structure for representations

- Deep learning relies heavily on Linear Algebra
 - Linear algebra uses tensors (e.g. 1-d tensor is a vector, 2-d tensor is a matrix)
 - PyTorch (and most deep learning frameworks), thus uses tensors (also called N-dimensional arrays).

What is a tensor?

- Two different understandings:
 - Generalization of vectors and matrices to an arbitrary number of dimensions
 - Multidimensional arrays

SCALAR $x[2]=5$	VECTOR $x[1, 0]=7$	MATRIX $x[0, 2, 1]=2$	TENSOR $x[1, 3, \dots, 2]=4$	TENSOR $x[1]$ N-D DATA → N INDICES
0D	1D	2D	3D	

Tensors are powerful for data representations

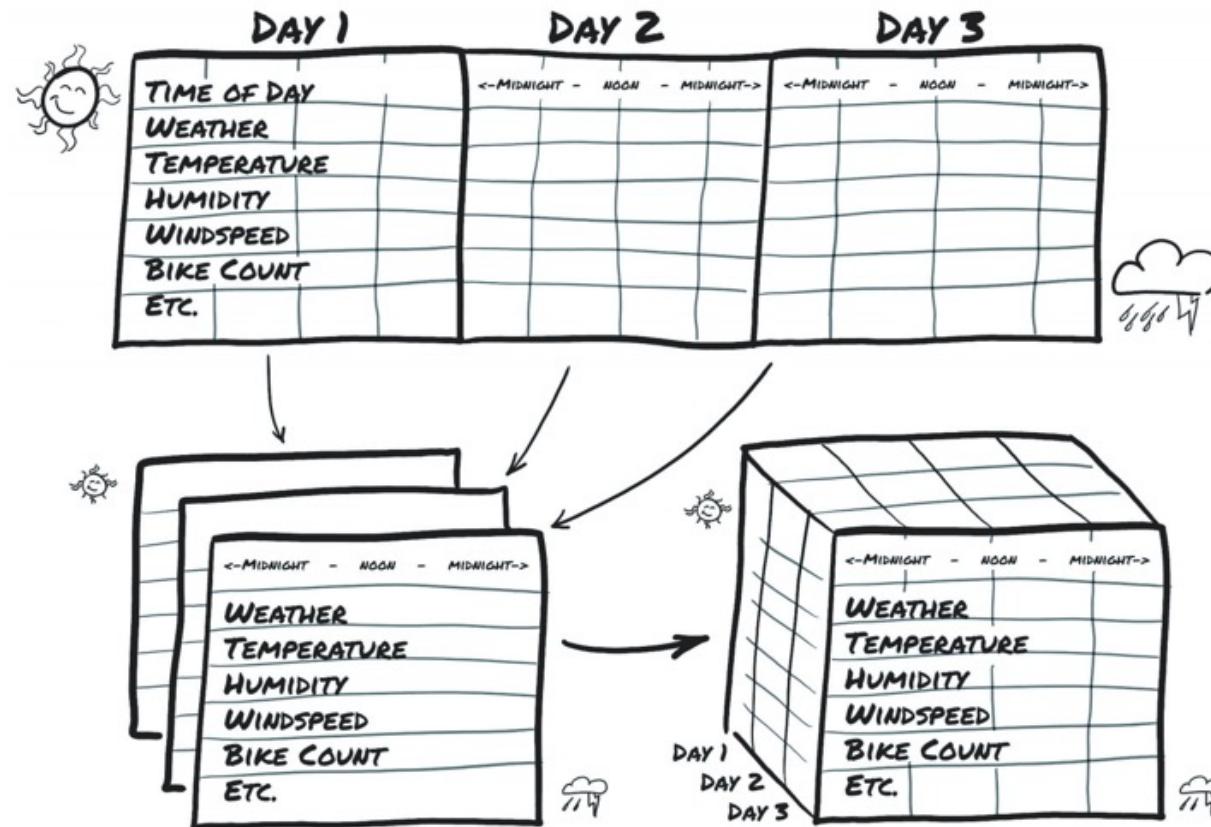
Examples

- 1. Tabular data:
 - Two-dimensional tensors (matrices)

Attributes of 2009EQ.csv Events								
Year	Month	Day	Time_UTC	Latitude	Longitude	Magnitude	Depth	Shape *
2009	1	1	12:43:12 PM	-15.43	-173.14	4	35	Point
2009	1	1	9:36:00 PM	17.22	40.52	5	10	Point
2009	1	1	9:50:24 AM	-55.16	-29.09	4.7	35	Point
2009	1	1	1:26:24 PM	80.85	-3.03	4.8	10	Point
2009	1	1	6:28:48 AM	-6.92	155.18	4.6	82	Point
2009	1	1	10:19:12 AM	-6.83	129.99	4.7	50	Point
2009	1	1	12:00:00 PM	-33.8	-72.72	4.6	0	Point
2009	1	1	12:57:36 PM	-58.29	-21.81	4.4	10	Point
2009	1	1	3:21:36 AM	-6.86	155.93	4.7	50	Point
2009	1	1	7:55:12 PM	1.12	120.73	4.7	49	Point
2009	1	1	11:16:48 AM	-11.66	166.75	4.7	254	Point
2009	1	1	2:09:36 AM	40.62	123.02	4.1	10	Point
2009	1	1	5:16:48 AM	-34.84	-107.65	5.8	10	Point
2009	1	1	1:40:48 PM	-9.61	120.72	4.2	20	Point
2009	1	1	2:38:24 AM	-22.04	-179.6	4.5	601	Point
2009	1	1	6:43:12 AM	1.32	121.84	5.1	33	Point
2009	1	1	4:19:12 PM	14.73	-91.39	4.7	169	Point
2009	1	1	5:45:36 PM	9.43	124.15	4.5	525	Point
2009	1	1	4:48:00 PM	-34.88	-107.78	5	10	Point
2009	1	1	8:09:36 PM	44.58	148.22	4.2	59	Point
2009	1	1	2:38:24 AM	-4.33	101.3	5.5	19	Point
2009	1	1	2:52:48 AM	-4.33	101.24	5.3	26	Point

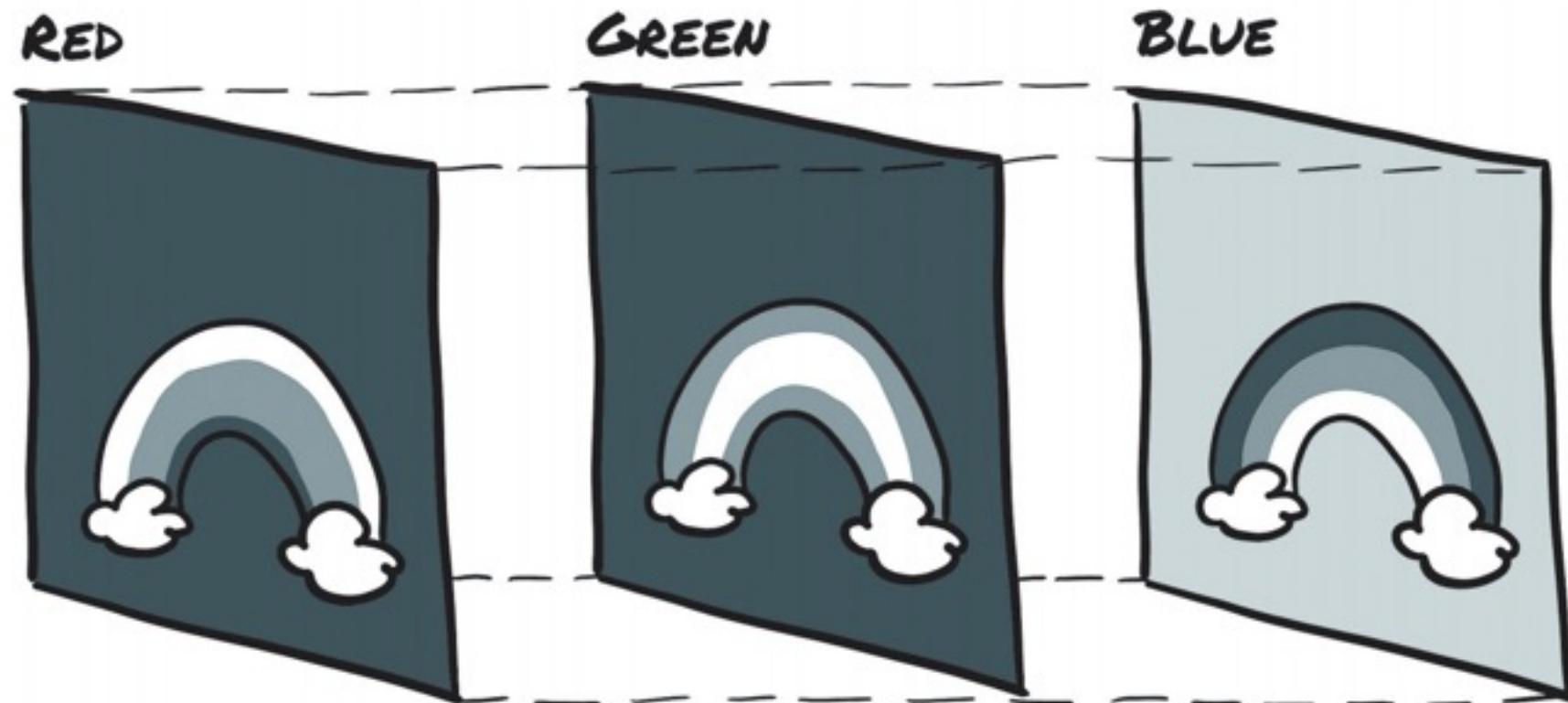
Tensors are powerful for data representations

- 2. Time-series data:
 - Three-dimensional tensors



Tensors are powerful for data representations

- 3. Images:
 - Three-dimensional tensors



Tensors are powerful for data representations

- 4. Texts:
 - As one-dimensional integer tensors
 - As two-dimensional float tensors (embeddings)



The fat cat sat on a mat
0 298 81 641 9 1 109



The fat cat sat on a mat
0.2 1.2 2.6
0.3 0.6 1.5
1.8 1.7 0.3

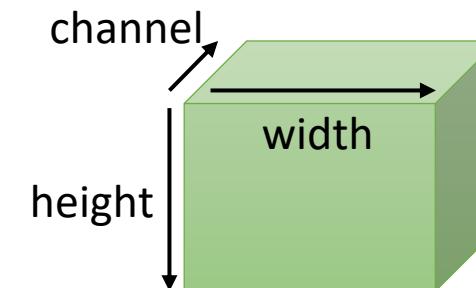
Tensors are powerful for data representations

- Tensors are powerful!
- Many different kinds of data can all be represented as tensors
- What kinds of operations do we have for tensors?

```
>>> from PIL import Image
```

Basic tensor operations

- A real example of images



```
!wget https://upload.wikimedia.org/wikipedia/en/7/7d/Lenna_%28test_image%29.png  
-O lenna.jpg  
>>> np_image = np.array(Image.open("lenna.png"))  
>>> image = torch.as_tensor(np_image)  
>>> plt.imshow(image)
```

Basic tensor operations

- 1. Tensor creation

- From Python lists or Numpy arrays

```
>>> torch.tensor([[0.1, 1.2], [2.2, 3.1], [4.9, 5.2]])
```

```
>>> torch.tensor(np.array([[0.1, 1.2], [2.2, 3.1], [4.9, 5.2]]))
```

- Tensors of a given size

```
>>> torch.tensor(2,3,4)
```

- Special tensors

```
>>> torch.zeros(2,3)
```

```
>>> torch.ones(2,3)
```

```
>>> torch.eye(3)
```

```
tensor([[ 0.,  0.,  0.],  
        [ 0.,  0.,  0.]])
```

```
tensor([[ 1.,  1.,  1.],  
        [ 1.,  1.,  1.]])
```

```
tensor([[ 1.,  0.,  0.],  
        [ 0.,  1.,  0.],  
        [ 0.,  0.,  1.]])
```

Basic tensor operations

- 2. Tensor properties
 - Shape

```
>>> x.shape
```

```
>>> x.size()
```

```
torch.Size([3, 2])
```

- Data type

```
>>> x.dtype
```

```
>>> x.type()
```

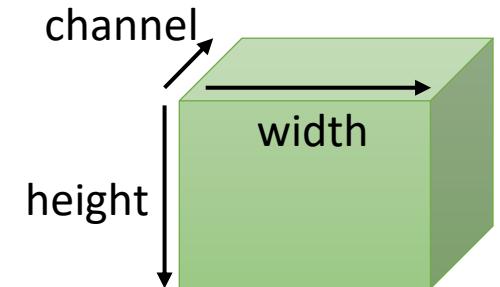
```
torch.float32
```

- Number of dimensions

```
>>> x.ndim
```

```
2
```

Basic tensor operations



- By convention
 - dimension refers to an axis of the tensor
 - size refers to the length of an axis in the tensor
 - index refers to a specific coordinate in the tensor

```
>>> print(image.shape)
>>> print(image.dtype)
>>> print(image.ndim)
```

```
torch.Size([512, 512, 3]) # (height, width, channel)
torch.uint8
3 # 3 dimensions: height, width, and channel
```

Basic tensor operations

- 3. Tensor type transformation

- Data type transformation

```
>>> x = x.int()
```

```
>>> x = x.float()
```

```
>>> x = x.double()
```

- Transformation between GPU and CPU

```
>>> x = x.cuda()
```

```
>>> x = x.cpu()
```

- Transform a tensor to Numpy arrays

```
>>> x = x.numpy()
```

```
>>> x = x.data.numpy()
```

Basic tensor operations

- 4. Tensor indexing

- Get an element

```
>>> x[0, 1]
```

```
tensor(2.)
```

- Get a row (the colon ":" stands for all elements)

```
>>> x[2, :]
```

```
tensor([5., 6.])
```

- Get a column

```
>>> x[:, 0]
```

```
tensor([1., 3., 5.])
```

- Get rows

```
>>> x[1:3, :]
```

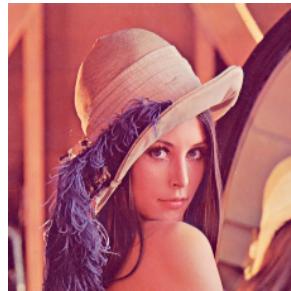
```
tensor([[3., 4.], [5., 6.]])
```

```
x = tensor([[ 1.,  2.],
            [ 3.,  4.],
            [ 5.,  6.]])
```

Basic tensor operations

- Operations on tensors are similar to their matrix counterparts

```
>>> plt.imshow(image[:, :256, :])
```



512×512×3



512×256×3

- Slice over the second axis (width axis)

For simplicity, the channel axis is not plotted here.

Basic tensor operations

- 5. Changing tensor dimensions

- Tensor reshaping

```
>>> x.reshape(6)
```

```
>>> x.view(6)
```

```
x = tensor([[ 1.,  2.],  
           [ 3.,  4.],  
           [ 5.,  6.]])
```

See example in Notebook

```
tensor([1., 2., 3., 4., 5., 6.])
```

- Tensor squeezing and unsqueezing

```
>>> torch.unsqueeze(x, 0)
```

```
>>> torch.squeeze(x, 0)
```

- Expansion

```
>>> x.expand(3, 2, 4)
```

```
>>> x.repeat(3, 2, 4)
```

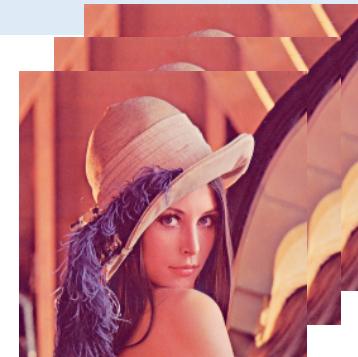
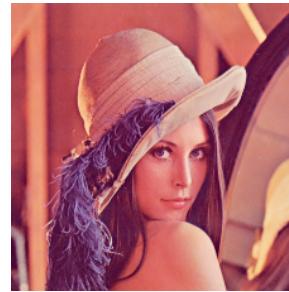
See example in Notebook

Basic tensor operations

- Extend a tensor

See example in Notebook

```
>>> batch = image.unsqueeze(0).repeat(3, 1, 1, 1)  
>>> plot(batch)
```



- Unsqueeze creates a new axis with size 1 at the specific dimension

For simplicity, the channel axis isn't plot here.

Basic tensor operations

```
x = tensor([1., 2., 3.])
```

- 6. Element-wise operations
 - Addition, subtraction, multiplication and division

```
>>> x + 3
```

```
tensor([4., 5., 6.])
```

- Exponential, logarithm, power

```
>>> x.exp()
```

```
tensor([ 2.7183, 7.3891, 20.0855])
```

```
>>> x.log()
```

```
tensor([ 0.0000, 0.6931, 1.0986])
```

```
>>> x.pow(2)
```

```
tensor([1., 4., 9.])
```

Basic tensor operations

- 6. A word about broadcasting
 - Addition, subtraction, multiplication and division

```
>>> x + 3
```

```
tensor([4., 5., 6.])
```

```
>>> x + x
```

```
tensor([2., 4., 6.])
```

- Both commands work, because PyTorch will try it's best to broadcast shapes for common operators such as addition (+), multiplication (*), etc

```
>>> y + x
```

```
Error, because the shapes (3,) and (2,2) cannot be broadcast
```

```
>>> y + z
```

```
tensor([[2., 4.],[4., 6.]])
```

```
>>> y + z.t()
```

```
tensor([[2., 3.],[5., 6.]])
```

```
x = tensor([1., 2., 3.])
```

```
y = tensor([[1., 2.],[3., 4.]])
```

```
z = tensor([[1., 2.]])
```

Basic tensor operations

- 7. Max/min/sum/mean
 - Overall max/min/sum/mean

```
>>> x.min()
```

```
tensor(1.)
```

```
>>> x.sum()
```

```
tensor(21.)
```

```
>>> x.mean()
```

```
tensor(3.5)
```

- Max/min/sum/mean on a specific axis

```
>>> x.sum(dim=0)
```

```
tensor([ 9., 12.])
```

```
>>> x.sum(dim=1)
```

```
tensor([ 3.,  7., 11.])
```

```
x = tensor([[ 1.,  2.],  
           [ 3.,  4.],  
           [ 5.,  6.]])
```

See example in Notebook

Basic tensor operations

- 8. Dot product and matrix multiplication

- Dot product

```
a = tensor([1., 2., 3.])
```

```
b = tensor([4., 5., 6.])
```

```
>>> torch.dot(a, b)
```

```
32
```

- Matrix multiplication

```
a = tensor([[ 1.,  2.],  
           [ 3.,  4.]])
```

```
b = tensor([[ 5.,  6.],  
           [ 7.,  8.]])
```

```
>>> torch.mm(a, b)
```

```
tensor([[19., 22.],  
       [43., 50.]])
```

Basic tensor operations

- 9. Commonly-used tensor operations in PyTorch

`torch.t`

`torch.transpose`

`torch.cat`

`torch.stack`

`torch.chunk`

`torch.unbind`

`torch.Tensor.view`

`torch.Tensor.reshape`

`torch.Tensor.expand`

`torch.squeeze`

`torch.unsqueeze`

`torch.min`

`torch.max`

`torch.sum`

`torch.mean`

`torch.eq`

`torch.ne`

`torch.mm`

`torch.bmm`

`torch.index_select`

`torch.masked_select`

`torch.Tensor.masked_fill_`

`torch.gather`

`torch.Tensor.scatter_`

Basic tensor operations

- Practice:
- **1. Softmax on a vector**
 - w is a vector of size d
 - $\text{softmax}(w)_i = \frac{e^{w_i}}{\sum_{k=1}^d e^{w_k}}$
- **2. KL divergence between two categorical distributions**
 - p and q are two d -dimensional categorical distributions
 - $KL(q, p) = \mathbb{E}_q \left[\log \frac{q}{p} \right] = \sum_x q(x) \log \frac{q(x)}{p(x)}$

Basic tensor operations

- Practice:
- 1. Softmax on a vector

```
>>> w.exp() / w.exp().sum()
```

- 2. KL divergence between two categorical distributions

```
>>> log_ratio = q.log() - p.log()  
>>> KL = torch.dot(q, log_ratio)
```

Training through auto-gradient

- How to train a deep learning model?

// Forward pass to make a prediction

- Convert input into floating-point numbers
- Use deep learning models to do transformation
 - A sequence of layers and intermediate representations
- Convert last representations into output

// Define a loss function

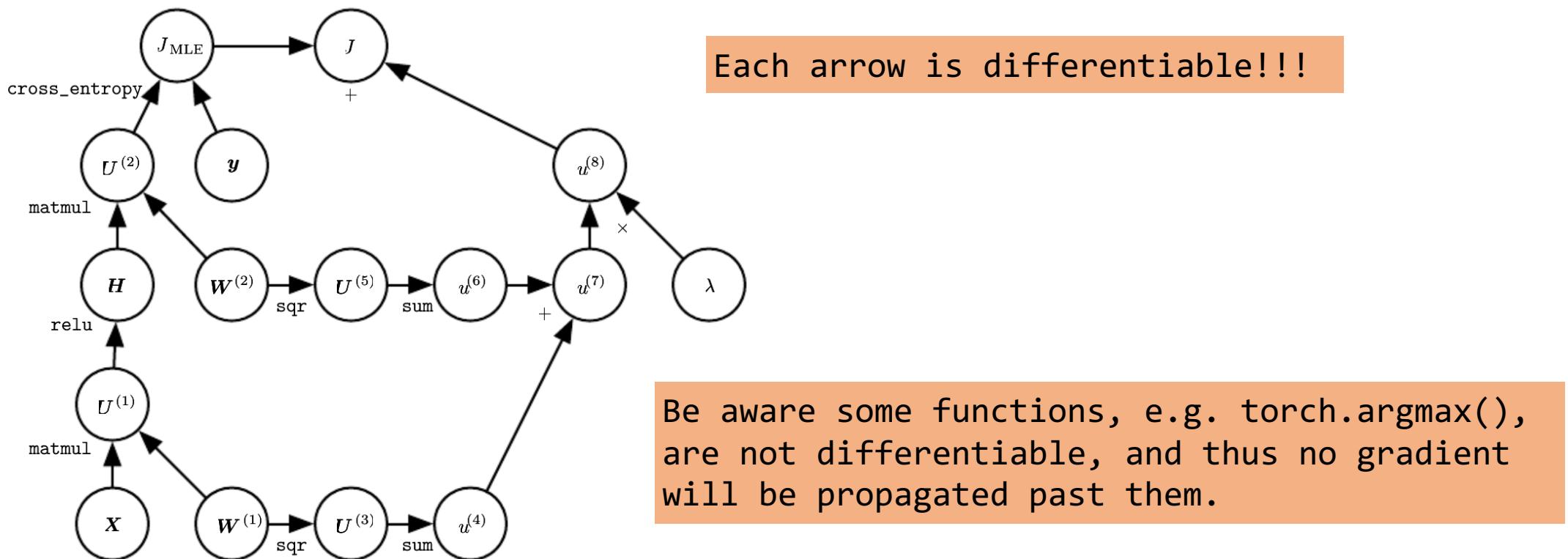
- Compute a scalar to measure the difference between predictions and targets

// Model learning

- Update model parameters

Training through auto-gradient

- Deep Learning book Chapter 6, computation graph example

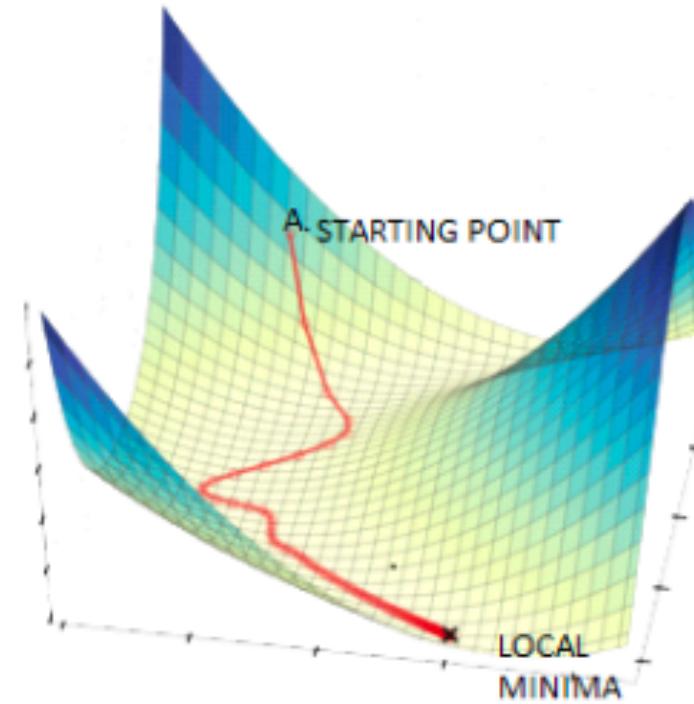
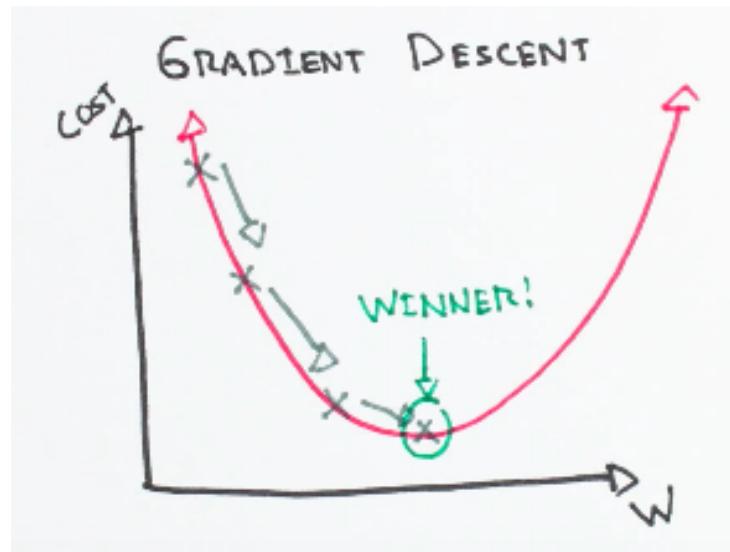


Training through auto-gradient

- How to update model parameters?

- Gradient descent

- $w^{(n+1)} = w^{(n)} - \epsilon \frac{\partial \mathcal{L}}{\partial w^{(n)}}$



Training through auto-gradient

- PyTorch's autograd: backpropagate all things

- Require gradients for a tensor

```
>>> w = torch.tensor([2.0], requires_grad=True)
```

- Compute a scalar loss \mathcal{L}

```
>>> L = f(w)
```

- Compute the gradient $\frac{\partial \mathcal{L}}{\partial w}$

```
>>> L.backward()
```

- Print the gradient

```
>>> print(w.grad)
```

- *Or compute using

```
>>> torch.autograd.grad(outputs=L, inputs=w)[0]
```

Training through auto-gradient

- Practice:
 - Compute the derivative of

$$f(x) = \exp(x^3 \sin(\log x)) \text{ at } x = 2$$

Training through auto-gradient

- Practice:
 - Compute the derivative of

$$f(x) = \exp(x^3 \sin(\log x)) \text{ at } x = 2$$

```
# Define the function f(x) = exp(x^3 * sin(log(x)))
L = torch.exp(w**3 * torch.sin(torch.log(w)))
# Perform backpropagation to compute the gradient by autograd
L.backward()
```

A toy example

- Linear classification:

- x is a 2-dimensional vector, $y \in \{0,1\}$ is its label.
- Learn a linear classifier to predict y given a x
- Forward pass to make a prediction

```
>>> pred = torch.mm(x, w) + b
```

- Compute a loss function

```
>>> loss = (pred - y).pow(2).mean()
```

- Update parameters

```
>>> loss.backward()  
>>> w.data -= w.grad * learning_rate  
>>> b.data -= b.grad * learning_rate
```

Summary of model training

- Forward pass to make a prediction
- Define a loss function
- Update model parameters

Summary of model training

- Forward pass to make a prediction

`torch.nn`

`torch.nn.functional`

- Define a loss function

`torch.nn` or `torch.nn.functional`

- Update model parameters

`torch.optim`

Building a Neural Network (general example)

```
>>> class ModelName(nn.Module):
>>>     def __init__(self, ModelHyperparameters):
>>>         super(ModelName, self).__init__()
>>>
>>>         self.layer = nn.Linear(input_dim, hidden_dim)
>>>         self.actv = nn.ReLU()
>>>         self.dropout = nn.Dropout(dropout)
>>>
>>>     def forward(self, X):
>>>         X = self.layer(X)
>>>         X = self.actv(X)
>>>         X = self.dropout(X)
>>>         return X
>>>
>>> model = ModelName(ModelHyperparameters)
>>> y = model(X) # X.shape == (batch_size, input_dim)
```

Building a Neural Network (general example)

- PyTorch has already coded many layers for us (some examples from `torch.nn`)

`torch.nn.Conv1d`

`torch.nn.MaxPool1d`

`torch.nn.RNNCell`

`torch.nn.Conv2d`

`torch.nn.MaxPool2d`

`torch.nn.GRUCell`

`torch.nn.Conv3d`

`torch.nn.MaxPool3d`

`torch.nn.LSTMCell`

`torch.nn.Linear`

`torch.nn.Embedding`

`torch.nn.BatchNorm1d`

`torch.nn.ReLU`

`torch.nn.Dropout`

`torch.nn.Softmax`

Training a Neural Network (general example)

```
>>> model = ModelName(ModelHyperparameters)
>>> optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
>>> for epoch in range(epochs):
>>>     model.train()
>>>     optimizer.zero_grad()
>>>     y = model(x)
>>>     loss = nn.functional.mse_loss(y, true_y)
>>>     loss.backward()
>>>     optimizer.step()
>>>     print('Epoch {}, Loss {:.4f}'.format(epoch, loss.item()))
```

Training a Neural Network (general example)

- PyTorch has already coded many loss functions and optimizers for us (some examples from `torch.nn.functional` (fn) and `torch.optim`)

`fn.mse_loss`

`torch.optim.Adam`

`fn.binary_cross_entropy`

`torch.optim.SGD`

`fn.kl_div`

`torch.optim.RMSprop`

`fn.nll_loss`

`torch.optim.LBFGS`

`fn.cross_entropy`

`torch.nn.cross_entropy_with_logits`

A digit recognition classifier

- First, let's download the dataset and create data loaders for them

```
>>> t = torchvision.transforms.ToTensor()  
>>> train = torchvision.datasets.MNIST("./data", train=True, download=True, transform=t)  
>>> test = torchvision.datasets.MNIST("./data", train=False, download=True, transform=t)  
  
>>> train_loader = torch.utils.data.DataLoader(train, batchsize=128, shuffle=True, ...)  
>>> test_loader = torch.utils.data.DataLoader(test, batchsize=128, shuffle=True, ...)
```

- And import necessary packages

```
>>> import torch  
>>> import torch.nn as nn  
>>> import torch.nn.functional as fn  
>>> import torchvision  
>>> import sklearn  
>>> import matplotlib.pyplot as plt
```

Neural Network

- The dataset, MNIST, is a classical dataset of handwritten digits.
- There are 10 classes and 70k B&W images.

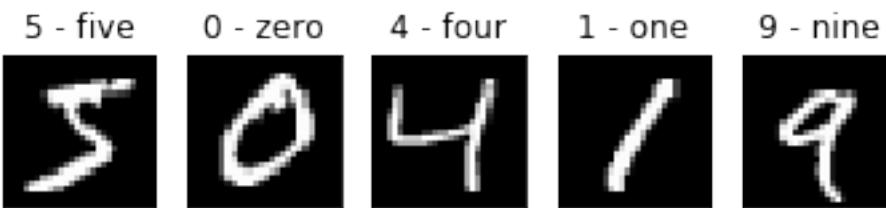


A digit recognition classifier

```
>>> def plot(data, labels, num_sample=5):  
>>>     n = min(len(data), num_sample)  
>>>     for i in range(n):  
>>>         plt.subplot(1, n, i+1)  
>>>         plt.imshow(data[i])  
>>>         plt.xticks([])  
>>>         plt.yticks([])  
>>>         plt.title(labels[i])
```

- To have some intuition, we can visualize a few samples by

```
>>> train.labels = [train.classes[target] for target in train.targets]  
>>> plot(train.data, train.labels)
```



A digit recognition classifier

- Now we can define our models
- We start from a simple multi-layer perceptron (MLP) model
 - ATM, you can just copy the code here :)

```
>>> class MLP(nn.Module):
>>>     def __init__(self, input_dim=784, hidden_dim=128, output_dim=10, dropout=0.5):
>>>         super(MLP, self).__init__()
>>>         self.fc1 = nn.Linear(input_dim, hidden_dim)
>>>         self.actv1 = nn.ReLU()
>>>         self.fc2 = nn.Linear(hidden_dim, output_dim)
>>>         self.actv2 = nn.Softmax(dim=-1)
>>>         self.dropout = nn.Dropout(dropout)
>>>
>>>     def forward(self, X):
>>>         X = X.flatten(1)
>>>         X = self.actv1(self.fc1(X))
>>>         X = self.dropout(X)
>>>         X = self.actv2(self.fc2(X))
>>>         return X
```

A digit recognition classifier

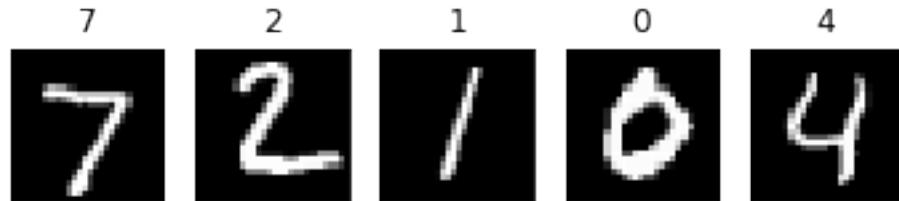
- Then we can train our model.

```
>>> optimizer = torch.optim.SGD(mlp.parameters(), lr=0.1)
>>> mlp = mlp.cuda() # bring the model to the gpu
>>> for epoch in range(20):
>>>     cur_loss = 0
>>>     mlp.train() # this ensures dropout and batch normalisation are activated
>>>     for (input, target) in train_loader:
>>>         x, y = input.cuda(), target.cuda() # porting the data to the GPU
>>>         optimizer.zero_grad() # gradients are accumulated, thus we have to 0 them
>>>         pred = mlp(x) # model forward pass
>>>         loss = fn.nll_loss(pred, y) # computing loss
>>>         cur_loss += loss.item()
>>>         loss.backward() # computing gradient
>>>         optimizer.step() # updating the parameter tensors with this line
>>>         print("Epoch {} | Train Loss {:.4f}".format(
>>>             epoch, cur_loss/len(train_loader)
>>> ))
```

A digit recognition classifier

- Let's try to make some predictions with our trained model.

```
>>> test.mlp_predictions = torch.argmax(mlp(test.data.float() / 255.0), dim=-1)  
>>> plot(test.data, test.mlp_predictions)
```



- It looks pretty good!
- We can obtain a statistical result by

```
>>>(test.mlp_predictions==test.targets).long().sum()/test.targets.shape[0]
```

- This prints the average accuracy of our model on the test set.
- Generally, it should be > 90%.

A digit recognition classifier

```
>>> import pickle
```

- We can save our model by

```
>>> with open("MLP_trained.pkl", "wb") as fout:  
>>>     pickle.dump(mlp, fout)
```

- To load it back, use

```
>>> with open("MLP_trained.pkl", "rb") as fin:  
>>>     mlp = pickle.load(fin)
```

Break time! 😬



PyTorch Part II

PyTorch 101

0 PyTorch on CPU or GPU

- There are generally two types of devices to do deep learning
 - CPU
 - GPU
- Tensors can be transferred (also as tensor operation) between them
- Note that any calculation requires tensors **on the exact SAME device**
- Practice in your Colab!

Content

- 1 General Pipeline: main components
- 2 Example 1: Iris Classification
 - 2.1 Data Preparation
 - 2.2 Try Linear Model + Gradient Descent
 - 2.3 Add Customized Model
 - 2.4 Add Mini-batch (Stochastic) Gradient Descent
- 3 Example 2: Image Classification
- 4 Debugging suggestions

1 General Pipeline

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

1 General Pipeline

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

```
>>> class Dataset(data.Dataset):  
>>>     def __init__(self, some_parameter):  
>>>         super(Model, self).__init__()  
>>>         self.X = ...  
>>>         self.y = ...  
>>>  
>>>     def __len__(self):  
>>>         return len(self.y)  
>>>  
>>>     def __getitem__(self, index):  
>>>         x_sample = self.X[index]  
>>>         y_sample = self.y[index]  
>>>         return x_sample, y_sample
```

1 General Pipeline

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

```
>>> class Model(nn.Module):
>>>     def __init__(self, some_parameter):
>>>         super(Model, self).__init__()
>>>         self.function_1 = ...
>>>         self.function_2 = ...
>>>         self.function_3 = ...
>>>
>>>     def forward(self, x):
>>>         intermediate_1 = self.function_1(x)
>>>         intermediate_2 = self.function_2(intermediate_1)
>>>         output = self.function_3(intermediate2)
>>>         return output
```

```
y = model(x)
y = model.forward(x)
```

1 General Pipeline

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

```
>>> dataset = Dataset()
>>> train_dataloader = data.DataLoader(dataset, batch_size=10, ...)
>>> model = Model()
>>> optimizer = optim.SGD(model.parameters(), lr=0.001)
>>> loss_function = nn.CrossEntropyLoss()
>>>
>>> for e in range(100):
>>>     for batch in train_dataloader:
>>>         X, y = batch
>>>         y_pred = model(X) # y_pred = model.forward(X)
>>>         loss = loss_function(y_pred, y_label)
>>>         loss.backward()
```

1 General Pipeline

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

```
>>> dataset = Dataset()
>>> test_dataloader = data.DataLoader(dataset, batch_size=10, ...)
>>>
>>> for batch in test_dataloader:
>>>     X, y = batch
>>>     y_pred = model(X)
>>>     acc = calculate_accuracy(y_pred, y)
```

2 Example: Iris Classification

2.1 Data preparation

2.2 Linear model + gradient descent

- Tip 1: More optimization methods and loss functions

2.3 Customized model

- Tip 2: More building blocks

2.4 Mini-batch (stochastic) gradient descent

2 Example: Iris Classification

2.1 Step 1: data preparation

Iris: 150 samples

- each has four features (sepal length, sepal width, petal length, petal width)
- each belongs to one of the three classes/types of Iris plant (Setosa, Versicolour, Virginica)

2.1 Data preparation

Iris: 150 samples

- each has four features (sepal length, sepal width, petal length, petal width)
- each belongs to one of the three classes/types of Iris plant (Setosa, Versicolour, Virginica)

x	y
[5.1 3.5 1.4 0.2]	0
[7. 3.2 4.7 1.4]	1
[6.3 3.3 6. 2.5]	2

2.1 Data preparation

```
>>> from sklearn import datasets
>>> from sklearn.model_selection import train_test_split
>>>
>>> def load_data():
>>>     iris = datasets.load_iris()
>>>     X = iris.data      # 150 * 4
>>>     y = iris.target    # 150
>>>     return X, y
>>>
>>> X, y = load_data()
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
>>>
>>> device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
>>> X_train = torch.FloatTensor(X_train).to(device)    # 120 * 4
>>> X_test = torch.FloatTensor(X_test).to(device)      # 30 * 4
>>> y_train = torch.LongTensor(y_train).to(device)      # 120
>>> y_test = torch.LongTensor(y_test).to(device)        # 30
```

2.2 Linear model + gradient descent

- Linear model:

$$\hat{y} = W^T X$$

- Loss function:

$$\ell(\hat{y}, y) = \text{Cross-Entropy}(\hat{y}, y)$$

- Gradient descent:

$$W^{t+1} = W^t - \eta \nabla_{W^t} \ell(\hat{y}, y)$$

$$= W^t - \frac{\eta}{N} \sum_{i=1}^N \nabla_{W^t} \ell(\hat{y}_i, y_i), \text{ where } N \text{ is the dataset size}$$

2.2 Linear model + gradient descent

```
>>> model = nn.Linear(4, 3).to(device)
>>>
>>> optimizer = optim.SGD(model.parameters(), lr=0.001)
>>> criterion = nn.CrossEntropyLoss()
>>>
>>> ##### Training #####
>>> model.train()
>>> for e in range(200):
>>>     ##### Get prediction #####
>>>     y_train_pred = model(X_train)
>>>     loss = criterion(y_train_pred, y_train)
>>>     ##### Clean-up gradients from previous steps #####
>>>     optimizer.zero_grad()
>>>     ##### Calculate gradients for current step #####
>>>     loss.backward()
>>>     ##### Update weights using SGD for current step #####
>>>     optimizer.step()
>>>     print('Epoch: {} \t Loss: {:.5f}'.format(e, loss.item()))
```

2.2 Linear model + gradient descent

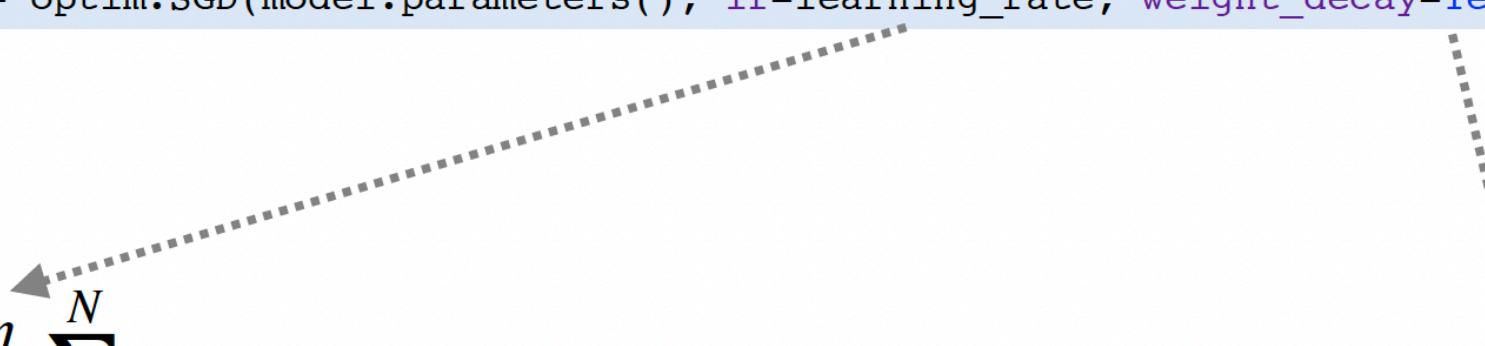
```
>>> ##### Evaluation #####
>>> model.eval()
>>> y_test_pred_output = model(X_test)
>>> _, y_test_pred = torch.max(y_test_pred_output, 1)
>>> acc = torch.true_divide(torch.sum(y_test_pred == y_test), y_test_pred.size()[0])
>>> print('accuracy: {}'.format(acc))
```

2.2 Linear model + gradient descent

- Tip 1: More optimization methods
- Common arguments:

```
>>> optimizer = optim.SGD(model.parameters(), lr=learning_rate, weight_decay=1e-5)
```

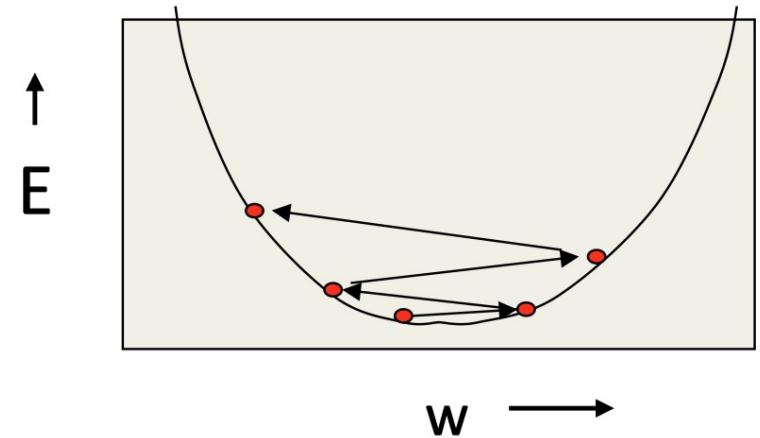
Recall:

$$W^{t+1} = W^t - \frac{\eta}{N} \sum_{i=1}^N \nabla_{W^t} \ell(\hat{y}_i, y_i), \text{ where } \ell(\hat{y}, y) = \text{Cross-Entropy}(\hat{y}, y) + \lambda \|W\|_2^2$$


2.2 Linear model + gradient descent

- Tip 1: More optimization methods
- A wide range of optimizers
 - SGD: the classical optimizer
 - RMSprop: a self-adaptive optimizer
 - Adam: a self-adaptive optimizer

adaptive gradient and lr



2.2 Linear model + gradient descent

- Tip 1: More loss functions

- Classification

`nn.NLLLoss`

$$\mathcal{L} = -\log(\text{pred}_{\text{target}})$$

`nn.CrossEntropyLoss`

$$\mathcal{L} = -\log(\text{softmax}(\text{pred})_{\text{target}})$$

- Regression

`nn.MSELoss`

$$\mathcal{L} = (\text{pred} - \text{target})^2$$

`nn.SmoothL1Loss`

$$\mathcal{L} = \min\left((\text{pred} - \text{target})^2, |\text{pred} - \text{target}|\right)$$

- Comparison / Ranking

`nn.MarginRankingLoss`

$$\mathcal{L} = \max(\text{pos} - \text{neg} + \text{margin}, 0)$$

`nn.TripletMarginLoss`

$$\mathcal{L} = \max(\text{dist}(\text{anchor}, \text{pos}) - \text{dist}(\text{anchor}, \text{neg}) + \text{margin}, 0)$$

Loss functions

- Loss functions are also non-parametric layers in PyTorch
- **Caveat: the model's output should be consistent with the loss**
- For `nn.NLLLoss`, it takes a log-probability distribution as input
 - The last layer in the forward function should be `F.LogSoftmax`
- For `nn.CrossEntropyLoss`, it takes unbounded logits as input
 - The last layer in the forward function shouldn't be any activation function
- For regression losses, it takes unbounded values as input

2.2 Linear model + gradient descent

Limitation of gradient descent:

Our toy example (Iris dataset) is very small, and we can fit all the samples into GPU/CPU memory at once.

But what if we have millions of data samples?

Then we will use mini-batch gradient descent, as will be discussed in 2.4

2.3 Customize models

Before in 2.2:

```
model = nn.Linear(4, 3).to(device)
```

Now customize our own model:

```
>>> class MultiLayerPercptron(nn.Module):
>>>     def __init__(self):
>>>         super(MultiLayerPercptron, self).__init__()
>>>         self.fc1 = nn.Linear(4, 100)
>>>         self.fc2 = nn.Linear(100, 3)
>>>     return
>>>
>>>     def forward(self, x):
>>>         x1 = self.fc1(x)                  # (120, 4)    => (120, 100)
>>>         x2 = self.fc2(x1)                # (120, 100) => (120, 3)
>>>         return x2
>>>
>>> model = MultiLayerPercptron().to(device)
```

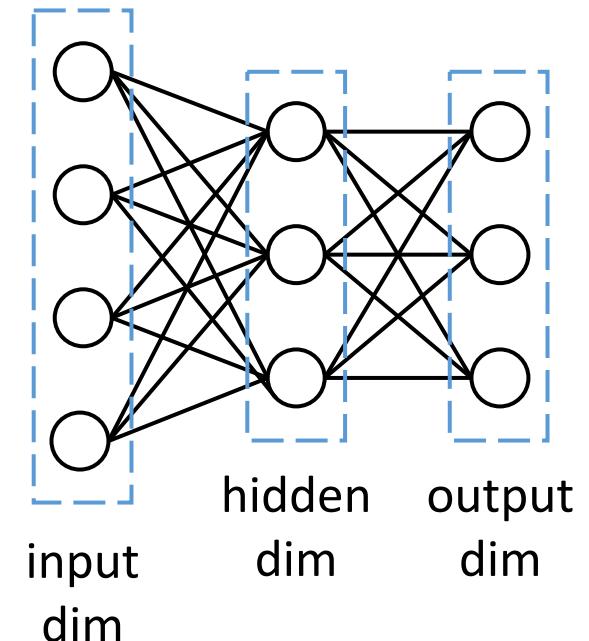
2.3 Customize models

- Basic receipt for customizing a model
 1. Define the parameters
 2. Define the forward function
 3. ~~Define the backward function~~
- Step 1 let the framework know **what to train.**
- Step 2 let the framework know **what the model is.**
- DL frameworks will automatically infer step 3 from step 2 (aka. autograd)

2.3 Customize models

- In the MLP example, the parameters are two linear layers

```
>>> class MLP(nn.Module):  
>>>     def __init__(self, input_dim, hidden_dim,  
output_dim):  
>>>         super(MLP, self).__init__()  
>>>         self.fc1 = nn.Linear(input_dim, hidden_dim)  
>>>         self.fc2 = nn.Linear(hidden_dim, output_dim)
```



- Here `nn.Linear` is a convenient interface to define all the parameters within a linear layer

Customize models

- Let's see how to put these ingredients into implementation
 - Inherit a class from nn.Module
 - Parameters are defined in `__init__()`
 - Forward function is defined as `forward()`

```
>>> class MyModel(nn.Module):
>>>     def __init__(self, ...):
>>>         super(MLP, self).__init__()
>>>         # here are parameter definitions
>>>         self.xxx = ...
>>>
>>>     def forward(self, ...):
>>>         # here is the forward function
>>>         return ...
```

```
>>> import torch.nn.functional as F
```

Customize models

- Forward function of MLP
- Very similar to NumPy

```
>>> class MLP(nn.Module):  
>>>     def forward(self, input):  
>>>         input = input.flatten(1)  
>>>         hidden = F.relu(self.fc1(input))  
>>>         output = F.softmax(self.fc2(hidden), dim=-1)  
>>>         return output
```

- `self.fc1` and `self.fc2` are called as functions, i.e. linear transformation
- `F.relu` and `F.softmax` are non-parameteric functions
 - They have no trainable parameters
 - We don't need to define them in `__init__()`

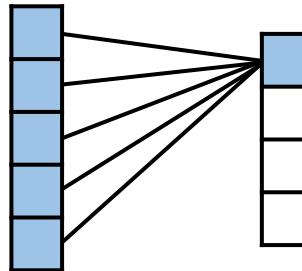
Customize models

- nn.Sequential is a convenient wrapper for multiple layers
- layers are applied in their definition order

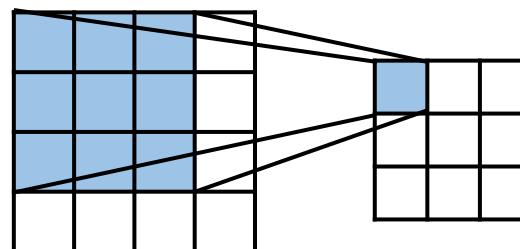
```
>>> class MLP(nn.Module):
>>>     def __init__(self, input_dim, hidden_dim, output_dim):
>>>         super(MLP, self).__init__()
>>>         self.model = nn.Sequential(
>                         nn.Linear(input_dim, hidden_dim),
>                         nn.Linear(hidden_dim, output_dim)
>                     )
>>>
>>>     def forward(self, input):
>>>         input = input.flatten(1)
>>>         output = F.softmax(self.model(input), dim=-1)
>>>         return output
```

Common building blocks

- Parametric layers
 - Linear layer (aka. fully connected / dense layer)
`nn.Linear(in_features, out_features, bias=True)`



- Convolution layer
`nn.Conv2d(in_channels, out_channels, kernel_size, stride=1)`

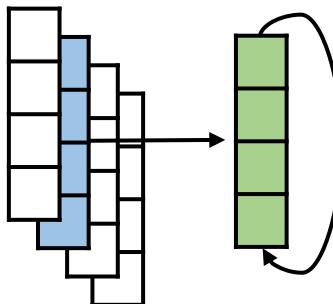


Common building blocks

- Parametric layers

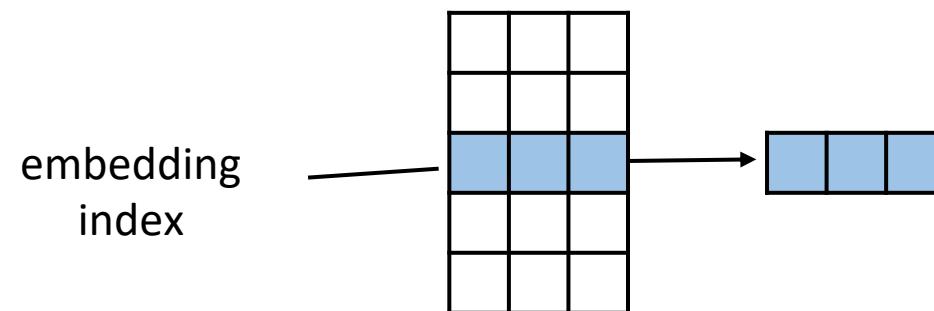
- Recurrent layer (multi-layer)

```
nn.LSTM(input_size, hidden_size, num_layers=1, bias=True)
```



- Embedding layer

```
nn.Embedding(num_embedding, embedding_dim, max_norm=None, norm_type=2.0)
```



Common building blocks

- Non-parametric layers

- Activation function

- `F.relu(input)`

- `F.sigmoid(input)`

- `F.tanh(input)`

- `F.softmax(input, dim=None)`

- Pooling function

- `F.avg_pool2d(kernel_size, stride=None)`

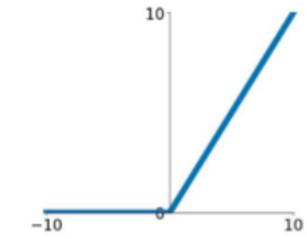
- `F.max_pool2d(kernel_size, stride=None)`

- Dropout layer

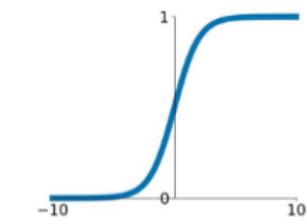
- `nn.Dropout(p=0.5)`

- ~~`F.dropout(input, p=0.5, training=True)`~~

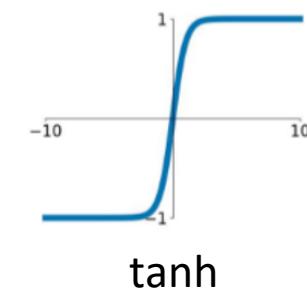
Not recommended!



ReLU



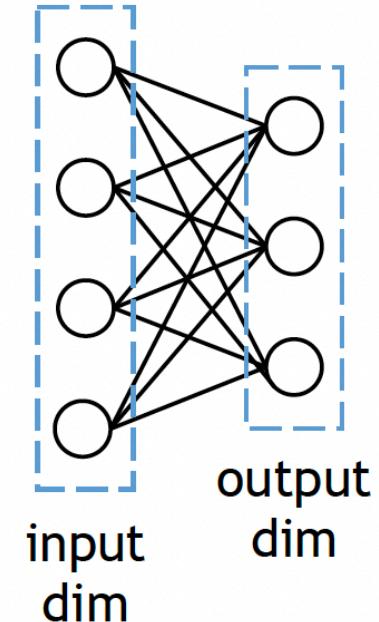
sigmoid



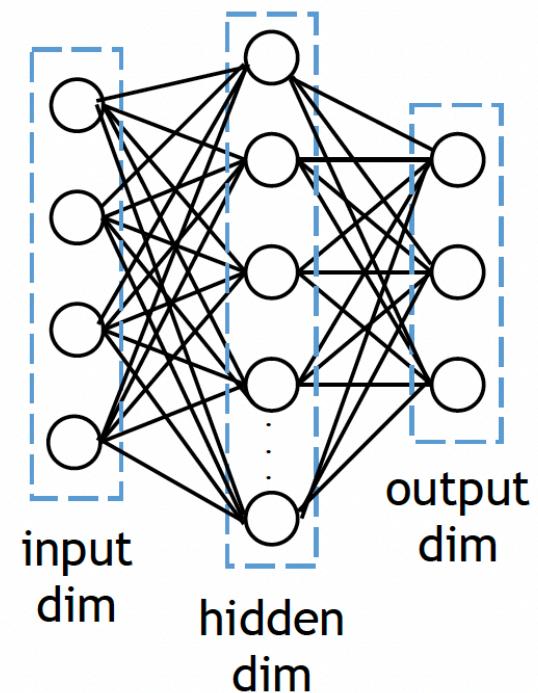
tanh

2.3 Customized model

```
model = nn.Linear(3, 4).to(device)
```



```
model = MLP().to(device)
```



2.4 Mini-Batch (stochastic) gradient descent

- Customized model:

$$\hat{y} = f_W(X)$$

- Loss function:

$$\ell(\hat{y}, y) = \text{Cross-Entropy}(\hat{y}, y)$$

- As mentioned at step 2, gradient descent has some limitations when it comes to larger dataset.

Solution: mini-batch (stochastic) gradient descent

$$\begin{aligned} W^{t+1} &= W^t - \eta \nabla_{W^t} \ell(\hat{y}, y) \\ &= W^t - \frac{\eta}{B} \sum_{i=1}^B \nabla_{W^t} \ell(\hat{y}_i, y_i), \text{ where } B \text{ is the batch size} \end{aligned}$$

2.4 Mini-Batch (stochastic) gradient descent

(1) Wrap-up customized dataset with torch.utils.data.Dataset

```
>>> class IrisDataset(data.Dataset):
>>>     def __init__(self, X, y):
>>>         self.X = X
>>>         self.y = y
>>>         return
>>>
>>>     def __len__(self):
>>>         return len(self.X)
>>>
>>>     def __getitem__(self, index):
>>>         X_sample = torch.FloatTensor(self.X[index])
>>>         y_sample = torch.LongTensor([self.y[index]])
>>>         return X_sample, y_sample
>>>
>>> train_dataset = IrisDataset(X_train, y_train)
>>> test_dataset = IrisDataset(X_test, y_test)
```

2.4 Mini-Batch (stochastic) gradient descent

(2) Put dataset into `torch.utils.data.DataLoader`.

```
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=10, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=10, shuffle=False)
```

Once we have the dataloader, then we can iterate over the whole dataset batch by batch.

```
>>> for e in range(100):
>>>     for batch in dataloader:
>>>         x_batch, y_batch = batch
>>>         ....
```

Recall in GD, what we did is following:

```
>>> for e in range(100):
>>>     x_train, y_train = ...
>>>     ....
```

2.4 Mini-Batch (stochastic) gradient descent

(2) Put dataset into `torch.utils.data.DataLoader`.

```
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=10, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=10, shuffle=False)
>>>
>>> for e in range(100):
>>>     for batch in dataloader:
>>>         x, y = batch
>>>         ....
```

shuffle=False, batch_size=10

e=0: (0, 1, ..., 9), (10, 11, ..., 19), ...

e=1: (0, 1, ..., 9), (10, 11, ..., 19), ...

e=2: (0, 1, ..., 9), (10, 11, ..., 19), ...

2.4 Mini-Batch (stochastic) gradient descent

(2) Put dataset into `torch.utils.data.DataLoader`.

```
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=10, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=10, shuffle=False)
>>>
>>> for e in range(100):
>>>     for batch in dataloader:
>>>         X, y = batch
>>>         ....
```

shuffle=True, batch_size=10

e=0: (29, 11, ..., 93), (3, 50, ..., 63), ...

e=1: (72, 23, ..., 18), (1, 31, ..., 60), ...

e=2: (45, 6, ..., 89), (2, 18, ..., 102), ...

2.4 Mini-Batch (stochastic) gradient descent

(2) Put dataset into `torch.utils.data.DataLoader`.

```
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=10, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=10, shuffle=False)
>>>
>>> for e in range(100):
>>>     for batch in dataloader:
>>>         X, y = batch
>>>         .....
```

Why shuffling?

- (1) Theoretical analysis: shuffling has smaller error upper bound, check this [paper](#).
- (2) Empirically, shuffling works quite well.
- (3) Intuitively, shuffling
 - (a) better matches with the independent and identical distribution (IID) assumption in most of the ML setting.
 - (b) makes the training harder, thus the learned model is more robust w.r.t. generalization performance.

2.4 Mini-Batch (stochastic) gradient descent

(3) Iterate over the train dataloader // for training

```
>>> model.train()
>>> for e in range(200):
>>>     accum_loss = 0
>>>     for batch in train_dataloader:
>>>         X_train_batch, y_train_batch = batch
>>>         X_train_batch = X_train_batch.to(device) # size: (batch_size, 4)
>>>         y_train_batch = y_train_batch.to(device) # size: (batch_size, 1)
>>>         y_train_batch = y_train_batch.squeeze(1) # size: (batch_size)
>>>         y_train_batch_pred = model(X_train_batch)
>>>         loss = criterion(y_train_batch_pred, y_train_batch)
>>>         optimizer.zero_grad()
>>>         loss.backward()
>>>         optimizer.step()
>>>         accum_loss += loss.item()
>>>     print('Epoch: {} \tLoss: {:.5f}'.format(e, accum_loss/len(train_dataloader)))
```

2.4 Mini-Batch (stochastic) gradient descent

(4) Iterate over the test dataloader // for evaluation

```
>>> model.eval()
>>> y_test, y_test_pred = [], []
>>> for batch in test_dataloader:
>>>     X_test_batch, y_test_batch = batch
>>>     X_test_batch = X_test_batch.to(device)      # size: (batch_size, 4)
>>>     y_test_batch = y_test_batch.to(device)      # size: (batch_size, 1)
>>>     y_test.append(y_test_batch)
>>>     y_test_pred_batch = model(X_test_batch)    # size: (batch_size, 1)
>>>     y_test_pred.append(y_test_pred_batch)
>>>
>>> y_test = torch.cat(y_test, dim=0)            # size: (30, 1)
>>> y_test = y_test.squeeze(1)                  # size: (30)
>>> y_test_pred = torch.cat(y_test_pred, dim=0)  # (30, 3)
>>> _, y_test_pred = torch.max(y_test_pred, 1)
>>> acc = torch.true_divide(torch.sum(y_test_pred == y_test), y_test_pred.size()[0])
>>> print('accuracy: {}'.format(acc))
```

3 Example 2: Image Classification

- Same pipeline
- Different Dataset
- Different Model

General Pipeline (review)

- Dataset
- Model
- Train model (optimizer, loss function)
- Test/Evaluate model

3 Example 2: Image Classification

- Same pipeline
- Different Dataset
 - create customized dataset (2.4) or use datasets provided by open-source package (`torchvision`)
- Different Model

```
>>> train_dataset = torchvision.datasets.MNIST('../data', train=True, transform=transform)
>>> test_dataset = torchvision.datasets.MNIST('../data', train=False, transform=transform)
>>>
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=128, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
>>> train_dataset = torchvision.datasets.CIFAR10('../data', train=True, transform=transform)
>>> test_dataset = torchvision.datasets.CIFAR10('../data', train=False, transform=transform)
>>>
>>> train_dataloader = data.DataLoader(train_dataset, batch_size=128, shuffle=True)
>>> test_dataloader = data.DataLoader(test_dataset, batch_size=128, shuffle=False)
```

3 Example 2: Image Classification

- Same pipeline
- Different Dataset
- Different Model
 - create customized model (2.3) or use models provided by open-source package (`torchvision`)
 - TorchVision provides stable implementations of CNN

```
>>> import torchvision.models as models  
>>>  
>>> resnet18 = models.resnet18()  
>>> alexnet = models.alexnet()  
>>> vgg16 = models.vgg16()  
>>> squeezenet = models.squeezenet1_0()  
>>> densenet = models.densenet161()
```

Debug models

- *Only 10% of programming is coding. The other 90% is debugging.*
 - Would be better if we are aware of common mistakes!
- General suggestions
- Shape errors
- Model errors
- Model capacity
- Implementation details



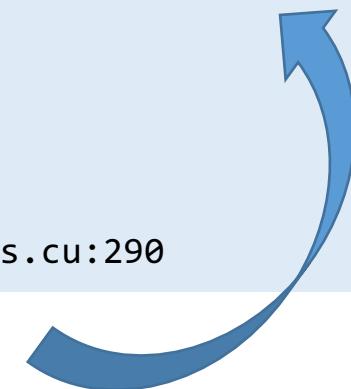
General suggestions

- Figure out where the bug is. A recommended order is
 - Check whether the code can run (-> e.g. shape errors)
 - Check the evaluation code
 - Check the ground truth
 - Check optimizer and learning rate
 - Check model errors
 - Check model capacity

Shape errors

- Shape errors are the most common reason if the code can't run

```
-----  
RuntimeError                                Traceback (most recent call last)  
<ipython-input-24-036a79cd99d7> in <module>()  
      6     device="cuda"  
      7 )  
----> 8 net.fit(train.data.to(torch.float32) / 255.0, train.targets)  
  
/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py in linear(input, weight, bias)  
 1368     if input.dim() == 2 and bias is not None:  
 1369         # fused op is marginally faster  
-> 1370         ret = torch.addmm(bias, input, weight.t())  
 1371     else:  
 1372         output = input.matmul(weight.t())  
  
RuntimeError: size mismatch, m1: [128 x 128], m2: [1 x 10] at THCTensorMathBlas.cu:290
```



- We can locate the error layer by the hint in the output

Shape errors

- Some typical shape errors

```
RuntimeError: Expected 4-dimensional input for 4-dimensional weight 64 3 7 7,  
but got 2-dimensional input of size [128, 2352] instead
```

- It means we have the wrong tensor dimension
- We should reshape the input with `tensor.view` or `tensor.reshape`

```
RuntimeError: size mismatch, m1: [128 x 784], m2: [128 x 10]
```

- It means we have the wrong size
- We should check both definitions of the layer and the input data shape

Model errors

- Model errors are unreasonable model design
 - They may cause phenomenon like gradient vanishing or gradient explosion
 - They can pass all assertions and thus are hard to find
- When shall we think of model errors?
 - Training diverge or converge badly
 - Tuning optimizer and learning rate doesn't help

Model errors

- Common reasons for model errors

- Consecutive transformation layers

```
x = self.fc2(self.fc1(x))
```

gradient explode

- Consecutive activation layers

```
x = F.sigmoid(F.sigmoid(x))
```

gradient vanish

- Typecast

```
x = torch.ones(..., dtype=torch.long)
```

unintended results

```
x = x / x.sum() # integer division
```



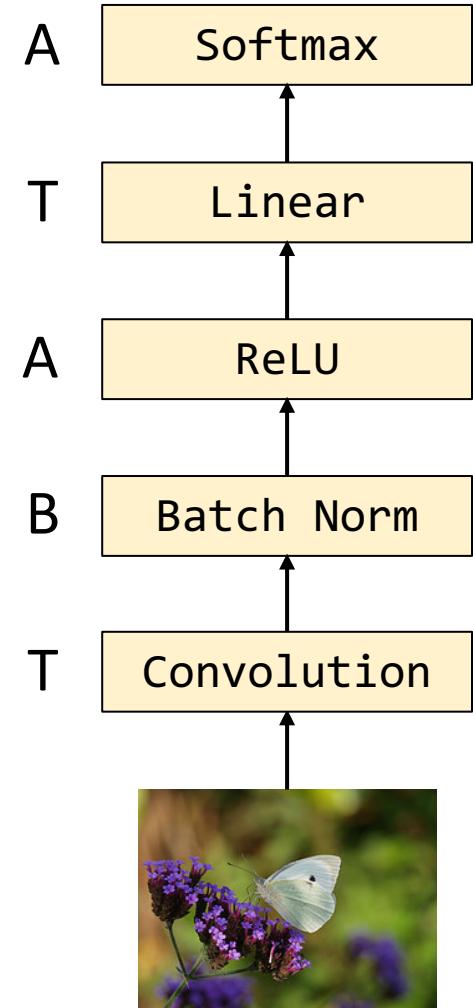
7 / 3 leads to 2 instead of 2.333

```
x = x / x.sum().float() # float division
```



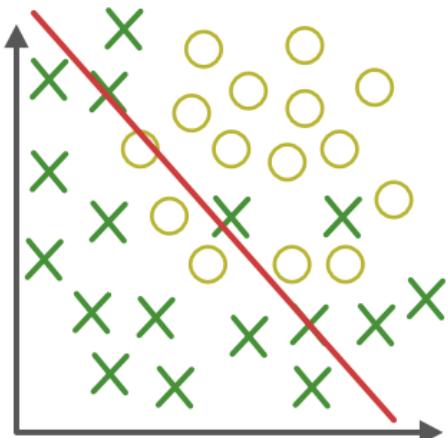
Model errors

- Common reasons for model errors
 - Incorrect position of normalization layers
 - Incorrect last activation layer
 -
- Too difficult to remember? Mnemonic: T(B)A
 - Transformation
 - Batch Normalization (optional)
 - Activation

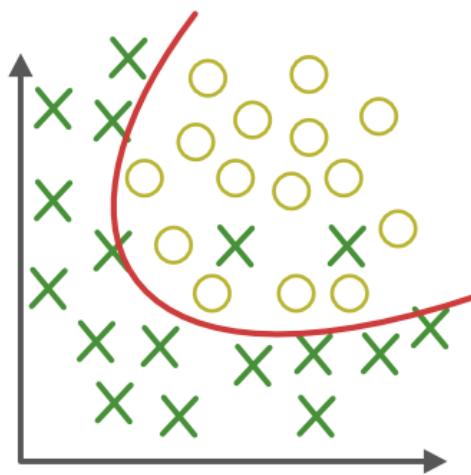


Model capacity

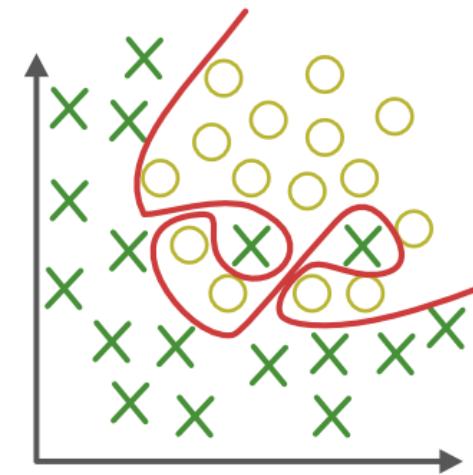
- Capacity: How powerful / complex a model is



Low capacity
Too simple to
explain the
observation



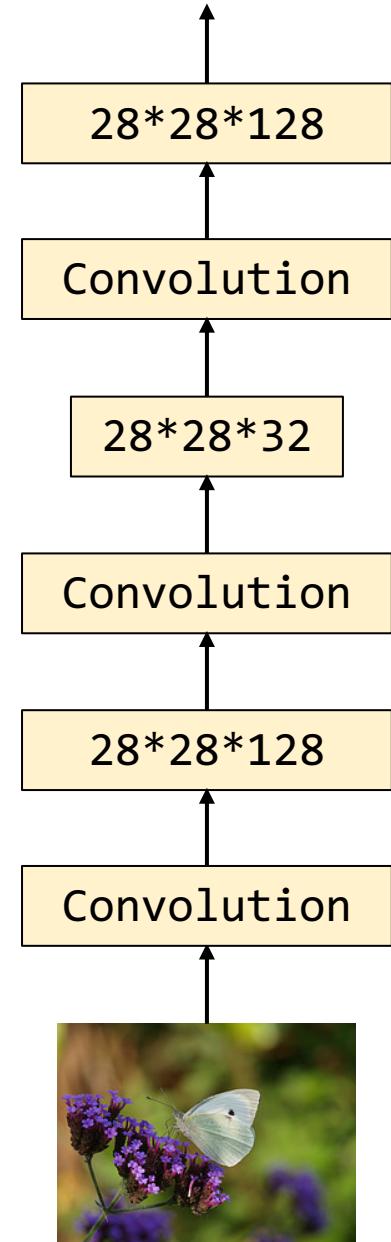
Appropriate capacity



High capacity
Too good to be
true

Model capacity

- Capacity is determined by
 - Model architecture
 - Number of learnable parameters
 - Regularization / Dropout / Early stopping
 -



Model capacity

- We can obtain #parameter by

```
>>> sum(np.prod(param.shape) for param in net.module_.parameters())
```

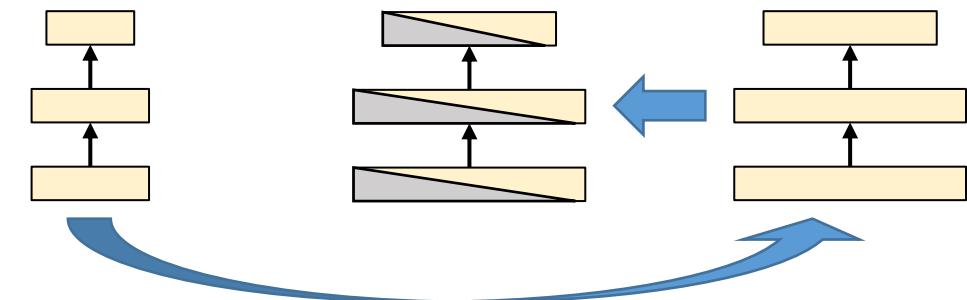
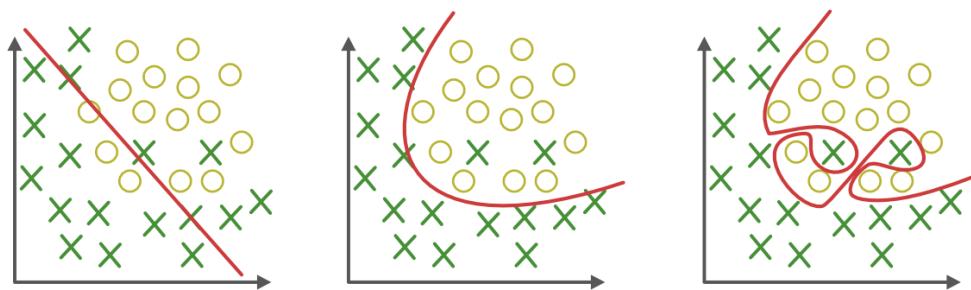
- ResNet18 has ~10M parameters
- GPT-3 has up to 175B parameters.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

- Comparing #parameter across different architectures may not be reliable

Model capacity

- Tips for tuning capacity. A recommend order is
 1. Choose an architecture
 2. Increase the number of hidden units if training is bad
 3. Add regularization if training is good but validation is bad



Other details

- **Normalizing the input** before you go
 - Usually it's better to use an input scale around 1

```
>>> model.fit(train.data.to(torch.float32) / 255.0, train.targets)
```
- Balance different categories (50% pos v.s. 50% neg)
 - Otherwise neural networks tend to guess the most frequent category
 - Like something what we do for multiple choice questions :)
 - A good practice is to reweight each category by the reciprocal of its frequency

Other details

- Model initialization
 - Implicitly carried out in any of these lines

```
>>> mlp = MLP()
```

```
>>> resnet18 = torchvision.models.resnet18()
```

- Remember to re-initialize our model every trial

Other details

- Random seed matters (sometimes)
 - Model initialization
 - Data loading order
- Some situations may not be reproduced when using a different random seed
- Fix a random seed

```
>>> seed = 123
>>> torch.random.manual_seed(seed)
>>> torch.cuda.manual_seed_all(seed)
```

Fast development: Rule of thumb

- Start with a small dataset and a short training epoch
 - Try different prototypes
 - Observe and find the best prototype
- Move to the full dataset
 - Try some variants of the best prototype
 - Find the best model
 - Increase to a long training epoch

Summary

- DL Frameworks are excellent helper for building own neural networks
- Tons of standard models / datasets are available in PyTorch
- GPU saves your training time by >1 magnitude
- Hyperparameter tuning is necessary for better performance
- Check(debug) details if powerful models don't work as expected
- Get more insights by playing with different models

Further readings

- Python / Numpy / Matplotlib tutorial
 - <http://cs231n.github.io/python-numpy-tutorial/>
- A simple neural network from scratch
 - <https://medium.com/dair-ai/a-simple-neural-network-from-scratch-with-pytorch-and-google-colab-c7f3830618e0>
- Language classification
 - https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/char_rnn_classification_tutorial.ipynb
- Dive into Deep Learning (PyTorch version)
 - <https://github.com/dsgiitr/d2l-pytorch>

Enjoy the PyTorch Journey!

