

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

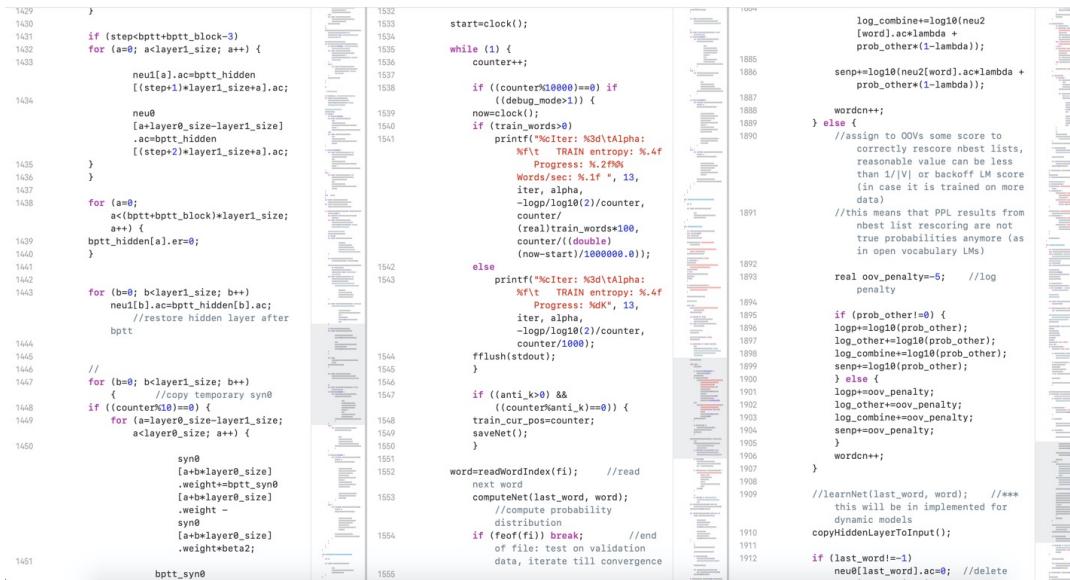
- CPU
 - Any language is applicable, Python, Matlab, R, C++, Java, etc.
 - Prefer languages with support to matrix(tensor) operation like Python, Matlab, R
 - Python has some nice packages/libraries
 - NumPy: matrix operation
 - Scikit-Learn: classic machine learning models
- GPU
 - Python has a large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

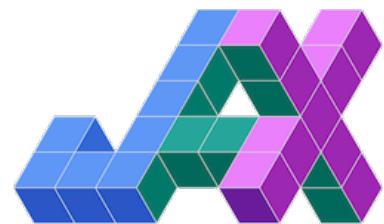


```
1429 }
1430 if (stepbptt+bptt_block>=3)
1431 for (a=0; a<layer1_size; a++) {
1432     neu[a].acbptt_hidden
1433     [(step+1)*layer1_size+a].ac;
1434
1435     neu# [a+layer0_size-layer1_size]
1436     .acbptt_hidden
1437     [(step+2)*layer1_size+a].ac;
1438 }
1439
1440 for (a=0;
1441     ac(bptt+bptt_block)*layer1_size;
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                     printf("%d\tAlpha: %f\t TRAIN entropy: %f\t
1540 Progress: %2.0f%\n", iter, alpha,
1541                     -log(log10(2))/counter,
1542                     counter/(real)train_words*100,
1543                     (now-start)/100000.0);
1544
1545                 else {
1546                     printf("%d\tAlpha: %f\t TRAIN entropy: %f\t
1547 Progress: %dK", iter,
1548                     alpha,
1549                     -log(log10(2))/counter,
1550                     counter/1000);
1551
1552                     ffflush(stdout);
1553
1554                     if ((anti_k>0) &
1555                         ((counter%anti_k)==0)) {
1556                         train.cur_pos=counter;
1557                         saveNet();
1558                     }
1559
1560                     word=readWordIndex(f1); //read
1561                     next word
1562                     computeNet(last_word, word);
1563                     //compute probability
1564                     distribution
1565                     if (eof(f1)) break; //end
1566                     of file: test on validation
1567                     data, iterate till convergence
1568
1569                     if (last_word!=~1)
1570                         neu# [last_word].ac=0; //delete
1571
1572                     wordcn++;
1573
1574                     log_combine+=log10(neu2
1575                     [word].ac*lambda +
1576                     prob_other*(1-lambda));
1577
1578                     senp+=log10(neu2[word].ac*lambda +
1579                     prob_other*(1-lambda));
1580
1581                     wordcn++;
1582
1583                     //assign to OOVs some score to
1584                     correctly rescore nbest lists,
1585                     reasonable value can be less
1586                     than 1/V or backoff LM score
1587                     (in case it is trained on more
1588                     data)
1589
1590                     //this means that PPL results from
1591                     nbest list rescoring are not
1592                     true probabilities anymore (as
1593                     in open vocabulary LMs)
1594
1595                     real oov_penalty=-5; //log
1596                     penalty
1597
1598                     if (prob_other!=0) {
1599                         logp+=log10(prob_other);
1600                         log_other+=log10(prob_other);
1601                         log_combine+=log10(prob_other);
1602                         senp+=log10(prob_other);
1603                     } else {
1604                         logp+=oov_penalty;
1605                         log_other+=oov_penalty;
1606                         log_combine+=oov_penalty;
1607                         senp+=oov_penalty;
1608                     }
1609                     wordcn++;
1610
1611                     if (prob_other!=0) {
1612                         logp+=log10(prob_other);
1613                         log_other+=log10(prob_other);
1614                         log_combine+=log10(prob_other);
1615                         senp+=log10(prob_other);
1616                     } else {
1617                         logp+=oov_penalty;
1618                         log_other+=oov_penalty;
1619                         log_combine+=oov_penalty;
1620                         senp+=oov_penalty;
1621                     }
1622
1623                     if (prob_other!=0) {
1624                         logp+=log10(prob_other);
1625                         log_other+=log10(prob_other);
1626                         log_combine+=log10(prob_other);
1627                         senp+=log10(prob_other);
1628                     } else {
1629                         logp+=oov_penalty;
1630                         log_other+=oov_penalty;
1631                         log_combine+=oov_penalty;
1632                         senp+=oov_penalty;
1633                     }
1634
1635                     if (prob_other!=0) {
1636                         logp+=log10(prob_other);
1637                         log_other+=log10(prob_other);
1638                         log_combine+=log10(prob_other);
1639                         senp+=log10(prob_other);
1640                     } else {
1641                         logp+=oov_penalty;
1642                         log_other+=oov_penalty;
1643                         log_combine+=oov_penalty;
1644                         senp+=oov_penalty;
1645                     }
1646
1647                     if (prob_other!=0) {
1648                         logp+=log10(prob_other);
1649                         log_other+=log10(prob_other);
1650                         log_combine+=log10(prob_other);
1651                         senp+=log10(prob_other);
1652
1653                     }
1654
1655                     if (prob_other!=0) {
1656                         logp+=log10(prob_other);
1657                         log_other+=log10(prob_other);
1658                         log_combine+=log10(prob_other);
1659                         senp+=log10(prob_other);
1660
1661                     }
1662
1663                     if (prob_other!=0) {
1664                         logp+=log10(prob_other);
1665                         log_other+=log10(prob_other);
1666                         log_combine+=log10(prob_other);
1667                         senp+=log10(prob_other);
1668
1669                     }
1670
1671                     if (prob_other!=0) {
1672                         logp+=log10(prob_other);
1673                         log_other+=log10(prob_other);
1674                         log_combine+=log10(prob_other);
1675                         senp+=log10(prob_other);
1676
1677                     }
1678
1679                     if (prob_other!=0) {
1680                         logp+=log10(prob_other);
1681                         log_other+=log10(prob_other);
1682                         log_combine+=log10(prob_other);
1683                         senp+=log10(prob_other);
1684
1685                     }
1686
1687                     if (prob_other!=0) {
1688                         logp+=log10(prob_other);
1689                         log_other+=log10(prob_other);
1690                         log_combine+=log10(prob_other);
1691                         senp+=log10(prob_other);
1692
1693                     }
1694
1695                     if (prob_other!=0) {
1696                         logp+=log10(prob_other);
1697                         log_other+=log10(prob_other);
1698                         log_combine+=log10(prob_other);
1699                         senp+=log10(prob_other);
1700
1701                     }
1702
1703                     if (prob_other!=0) {
1704                         logp+=log10(prob_other);
1705                         log_other+=log10(prob_other);
1706                         log_combine+=log10(prob_other);
1707                         senp+=log10(prob_other);
1708
1709                     }
1710
1711                     if (prob_other!=0) {
1712                         logp+=log10(prob_other);
1713                         log_other+=log10(prob_other);
1714                         log_combine+=log10(prob_other);
1715                         senp+=log10(prob_other);
1716
1717                     }
1718
1719                     if (prob_other!=0) {
1720                         logp+=log10(prob_other);
1721                         log_other+=log10(prob_other);
1722                         log_combine+=log10(prob_other);
1723                         senp+=log10(prob_other);
1724
1725                     }
1726
1727                     if (prob_other!=0) {
1728                         logp+=log10(prob_other);
1729                         log_other+=log10(prob_other);
1730                         log_combine+=log10(prob_other);
1731                         senp+=log10(prob_other);
1732
1733                     }
1734
1735                     if (prob_other!=0) {
1736                         logp+=log10(prob_other);
1737                         log_other+=log10(prob_other);
1738                         log_combine+=log10(prob_other);
1739                         senp+=log10(prob_other);
1740
1741                     }
1742
1743                     if (prob_other!=0) {
1744                         logp+=log10(prob_other);
1745                         log_other+=log10(prob_other);
1746                         log_combine+=log10(prob_other);
1747                         senp+=log10(prob_other);
1748
1749                     }
1750
1751                     if (prob_other!=0) {
1752                         logp+=log10(prob_other);
1753                         log_other+=log10(prob_other);
1754                         log_combine+=log10(prob_other);
1755                         senp+=log10(prob_other);
1756
1757                     }
1758
1759                     if (prob_other!=0) {
1760                         logp+=log10(prob_other);
1761                         log_other+=log10(prob_other);
1762                         log_combine+=log10(prob_other);
1763                         senp+=log10(prob_other);
1764
1765                     }
1766
1767                     if (prob_other!=0) {
1768                         logp+=log10(prob_other);
1769                         log_other+=log10(prob_other);
1770                         log_combine+=log10(prob_other);
1771                         senp+=log10(prob_other);
1772
1773                     }
1774
1775                     if (prob_other!=0) {
1776                         logp+=log10(prob_other);
1777                         log_other+=log10(prob_other);
1778                         log_combine+=log10(prob_other);
1779                         senp+=log10(prob_other);
1780
1781                     }
1782
1783                     if (prob_other!=0) {
1784                         logp+=log10(prob_other);
1785                         log_other+=log10(prob_other);
1786                         log_combine+=log10(prob_other);
1787                         senp+=log10(prob_other);
1788
1789                     }
1790
1791                     if (prob_other!=0) {
1792                         logp+=log10(prob_other);
1793                         log_other+=log10(prob_other);
1794                         log_combine+=log10(prob_other);
1795                         senp+=log10(prob_other);
1796
1797                     }
1798
1799                     if (prob_other!=0) {
1800                         logp+=log10(prob_other);
1801                         log_other+=log10(prob_other);
1802                         log_combine+=log10(prob_other);
1803                         senp+=log10(prob_other);
1804
1805                     }
1806
1807                     if (prob_other!=0) {
1808                         logp+=log10(prob_other);
1809                         log_other+=log10(prob_other);
1810                         log_combine+=log10(prob_other);
1811                         senp+=log10(prob_other);
1812
1813                     }
1814
1815                     if (prob_other!=0) {
1816                         logp+=log10(prob_other);
1817                         log_other+=log10(prob_other);
1818                         log_combine+=log10(prob_other);
1819                         senp+=log10(prob_other);
1820
1821                     }
1822
1823                     if (prob_other!=0) {
1824                         logp+=log10(prob_other);
1825                         log_other+=log10(prob_other);
1826                         log_combine+=log10(prob_other);
1827                         senp+=log10(prob_other);
1828
1829                     }
1830
1831                     if (prob_other!=0) {
1832                         logp+=log10(prob_other);
1833                         log_other+=log10(prob_other);
1834                         log_combine+=log10(prob_other);
1835                         senp+=log10(prob_other);
1836
1837                     }
1838
1839                     if (prob_other!=0) {
1840                         logp+=log10(prob_other);
1841                         log_other+=log10(prob_other);
1842                         log_combine+=log10(prob_other);
1843                         senp+=log10(prob_other);
1844
1845                     }
1846
1847                     if (prob_other!=0) {
1848                         logp+=log10(prob_other);
1849                         log_other+=log10(prob_other);
1850                         log_combine+=log10(prob_other);
1851                         senp+=log10(prob_other);
1852
1853                     }
1854
1855                     if (prob_other!=0) {
1856                         logp+=log10(prob_other);
1857                         log_other+=log10(prob_other);
1858                         log_combine+=log10(prob_other);
1859                         senp+=log10(prob_other);
1860
1861                     }
1862
1863                     if (prob_other!=0) {
1864                         logp+=log10(prob_other);
1865                         log_other+=log10(prob_other);
1866                         log_combine+=log10(prob_other);
1867                         senp+=log10(prob_other);
1868
1869                     }
1870
1871                     if (prob_other!=0) {
1872                         logp+=log10(prob_other);
1873                         log_other+=log10(prob_other);
1874                         log_combine+=log10(prob_other);
1875                         senp+=log10(prob_other);
1876
1877                     }
1878
1879                     if (prob_other!=0) {
1880                         logp+=log10(prob_other);
1881                         log_other+=log10(prob_other);
1882                         log_combine+=log10(prob_other);
1883                         senp+=log10(prob_other);
1884
1885                     }
1886
1887                     if (prob_other!=0) {
1888                         logp+=log10(prob_other);
1889                         log_other+=log10(prob_other);
1890                         log_combine+=log10(prob_other);
1891                         senp+=log10(prob_other);
1892
1893                     }
1894
1895                     if (prob_other!=0) {
1896                         logp+=log10(prob_other);
1897                         log_other+=log10(prob_other);
1898                         log_combine+=log10(prob_other);
1899                         senp+=log10(prob_other);
1900
1901                     }
1902
1903                     if (prob_other!=0) {
1904                         logp+=log10(prob_other);
1905                         log_other+=log10(prob_other);
1906                         log_combine+=log10(prob_other);
1907                         senp+=log10(prob_other);
1908
1909                     }
1910
1911                     if (prob_other!=0) {
1912                         logp+=log10(prob_other);
1913                         log_other+=log10(prob_other);
1914                         log_combine+=log10(prob_other);
1915                         senp+=log10(prob_other);
1916
1917                     }
1918
1919                     if (prob_other!=0) {
1920                         logp+=log10(prob_other);
1921                         log_other+=log10(prob_other);
1922                         log_combine+=log10(prob_other);
1923                         senp+=log10(prob_other);
1924
1925                     }
1926
1927                     if (prob_other!=0) {
1928                         logp+=log10(prob_other);
1929                         log_other+=log10(prob_other);
1930                         log_combine+=log10(prob_other);
1931                         senp+=log10(prob_other);
1932
1933                     }
1934
1935                     if (prob_other!=0) {
1936                         logp+=log10(prob_other);
1937                         log_other+=log10(prob_other);
1938                         log_combine+=log10(prob_other);
1939                         senp+=log10(prob_other);
1940
1941                     }
1942
1943                     if (prob_other!=0) {
1944                         logp+=log10(prob_other);
1945                         log_other+=log10(prob_other);
1946                         log_combine+=log10(prob_other);
1947                         senp+=log10(prob_other);
1948
1949                     }
1950
1951                     if (prob_other!=0) {
1952                         logp+=log10(prob_other);
1953                         log_other+=log10(prob_other);
1954                         log_combine+=log10(prob_other);
1955                         senp+=log10(prob_other);
1956
1957                     }
1958
1959                     if (prob_other!=0) {
1960                         logp+=log10(prob_other);
1961                         log_other+=log10(prob_other);
1962                         log_combine+=log10(prob_other);
1963                         senp+=log10(prob_other);
1964
1965                     }
1966
1967                     if (prob_other!=0) {
1968                         logp+=log10(prob_other);
1969                         log_other+=log10(prob_other);
1970                         log_combine+=log10(prob_other);
1971                         senp+=log10(prob_other);
1972
1973                     }
1974
1975                     if (prob_other!=0) {
1976                         logp+=log10(prob_other);
1977                         log_other+=log10(prob_other);
1978                         log_combine+=log10(prob_other);
1979                         senp+=log10(prob_other);
1980
1981                     }
1982
1983                     if (prob_other!=0) {
1984                         logp+=log10(prob_other);
1985                         log_other+=log10(prob_other);
1986                         log_combine+=log10(prob_other);
1987                         senp+=log10(prob_other);
1988
1989                     }
1990
1991                     if (prob_other!=0) {
1992                         logp+=log10(prob_other);
1993                         log_other+=log10(prob_other);
1994                         log_combine+=log10(prob_other);
1995                         senp+=log10(prob_other);
1996
1997                     }
1998
1999                     if (prob_other!=0) {
2000                         logp+=log10(prob_other);
2001                         log_other+=log10(prob_other);
2002                         log_combine+=log10(prob_other);
2003                         senp+=log10(prob_other);
2004
2005                     }
2006
2007                     if (prob_other!=0) {
2008                         logp+=log10(prob_other);
2009                         log_other+=log10(prob_other);
2010                         log_combine+=log10(prob_other);
2011                         senp+=log10(prob_other);
2012
2013                     }
2014
2015                     if (prob_other!=0) {
2016                         logp+=log10(prob_other);
2017                         log_other+=log10(prob_other);
2018                         log_combine+=log10(prob_other);
2019                         senp+=log10(prob_other);
2020
2021                     }
2022
2023                     if (prob_other!=0) {
2024                         logp+=log10(prob_other);
2025                         log_other+=log10(prob_other);
2026                         log_combine+=log10(prob_other);
2027                         senp+=log10(prob_other);
2028
2029                     }
2030
2031                     if (prob_other!=0) {
2032                         logp+=log10(prob_other);
2033                         log_other+=log10(prob_other);
2034                         log_combine+=log10(prob_other);
2035                         senp+=log10(prob_other);
2036
2037                     }
2038
2039                     if (prob_other!=0) {
2040                         logp+=log10(prob_other);
2041                         log_other+=log10(prob_other);
2042                         log_combine+=log10(prob_other);
2043                         senp+=log10(prob_other);
2044
2045                     }
2046
2047                     if (prob_other!=0) {
2048                         logp+=log10(prob_other);
2049                         log_other+=log10(prob_other);
2050                         log_combine+=log10(prob_other);
2051                         senp+=log10(prob_other);
2052
2053                     }
2054
2055                     if (prob_other!=0) {
2056                         logp+=log10(prob_other);
2057                         log_other+=log10(prob_other);
2058                         log_combine+=log10(prob_other);
2059                         senp+=log10(prob_other);
2060
2061                     }
2062
2063                     if (prob_other!=0) {
2064                         logp+=log10(prob_other);
2065                         log_other+=log10(prob_other);
2066                         log_combine+=log10(prob_other);
2067                         senp+=log10(prob_other);
2068
2069                     }
2070
2071                     if (prob_other!=0) {
2072                         logp+=log10(prob_other);
2073                         log_other+=log10(prob_other);
2074                         log_combine+=log10(prob_other);
2075                         senp+=log10(prob_other);
2076
2077                     }
2078
2079                     if (prob_other!=0) {
2080                         logp+=log10(prob_other);
2081                         log_other+=log10(prob_other);
2082                         log_combine+=log10(prob_other);
2083                         senp+=log10(prob_other);
2084
2085                     }
2086
2087                     if (prob_other!=0) {
2088                         logp+=log10(prob_other);
2089                         log_other+=log10(prob_other);
2090                         log_combine+=log10(prob_other);
2091                         senp+=log10(prob_other);
2092
2093                     }
2094
2095                     if (prob_other!=0) {
2096                         logp+=log10(prob_other);
2097                         log_other+=log10(prob_other);
2098                         log_combine+=log10(prob_other);
2099                         senp+=log10(prob_other);
2100
2101                     }
2102
2103                     if (prob_other!=0) {
2104                         logp+=log10(prob_other);
2105                         log_other+=log10(prob_other);
2106                         log_combine+=log10(prob_other);
2107                         senp+=log10(prob_other);
2108
2109                     }
2110
2111                     if (prob_other!=0) {
2112                         logp+=log10(prob_other);
2113                         log_other+=log10(prob_other);
2114                         log_combine+=log10(prob_other);
2115                         senp+=log10(prob_other);
2116
2117                     }
2118
2119                     if (prob_other!=0) {
2120                         logp+=log10(prob_other);
2121                         log_other+=log10(prob_other);
2122                         log_combine+=log10(prob_other);
2123                         senp+=log10(prob_other);
2124
2125                     }
2126
2127                     if (prob_other!=0) {
2128                         logp+=log10(prob_other);
2129                         log_other+=log10(prob_other);
2130                         log_combine+=log10(prob_other);
2131                         senp+=log10(prob_other);
2132
2133                     }
2134
2135                     if (prob_other!=0) {
2136                         logp+=log10(prob_other);
2137                         log_other+=log10(prob_other);
2138                         log_combine+=log10(prob_other);
2139                         senp+=log10(prob_other);
2140
2141                     }
2142
2143                     if (prob_other!=0) {
2144                         logp+=log10(prob_other);
2145                         log_other+=log10(prob_other);
2146                         log_combine+=log10(prob_other);
2147                         senp+=log10(prob_other);
2148
2149                     }
2150
2151                     if (prob_other!=0) {
2152                         logp+=log10(prob_other);
2153                         log_other+=log10(prob_other);
2154                         log_combine+=log10(prob_other);
2155                         senp+=log10(prob_other);
2156
2157                     }
2158
2159                     if (prob_other!=0) {
2160                         logp+=log10(prob_other);
2161                         log_other+=log10(prob_other);
2162                         log_combine+=log10(prob_other);
2163                         senp+=log10(prob_other);
2164
2165                     }
2166
2167                     if (prob_other!=0) {
2168                         logp+=log10(prob_other);
2169                         log_other+=log10(prob_other);
2170                         log_combine+=log10(prob_other);
2171                         senp+=log10(prob_other);
2172
2173                     }
2174
2175                     if (prob_other!=0) {
2176                         logp+=log10(prob_other);
2177                         log_other+=log10(prob_other);
2178                         log_combine+=log10(prob_other);
2179                         senp+=log10(prob_other);
2180
2181                     }
2182
2183                     if (prob_other!=0) {
2184                         logp+=log10(prob_other);
2185                         log_other+=log10(prob_other);
2186                         log_combine+=log10(prob_other);
2187                         senp+=log10(prob_other);
2188
2189                     }
2190
2191                     if (prob_other!=0) {
2192                         logp+=log10(prob_other);
2193                         log_other+=log10(prob_other);
2194                         log_combine+=log10(prob_other);
2195                         senp+=log10(prob_other);
2196
2197                     }
2198
2199                     if (prob_other!=0) {
2200                         logp+=log10(prob_other);
2201                         log_other+=log10(prob_other);
2202                         log_combine+=log10(prob_other);
2203                         senp+=log10(prob_other);
2204
2205                     }
2206
2207                     if (prob_other!=0) {
2208                         logp+=log10(prob_other);
2209                         log_other+=log10(prob_other);
2210                         log_combine+=log10(prob_other);
2211                         senp+=log10(prob_other);
2212
2213                     }
2214
2215                     if (prob_other!=0) {
2216                         logp+=log10(prob_other);
2217                         log_other+=log10(prob_other);
2218                         log_combine+=log10(prob_other);
2219                         senp+=log10(prob_other);
2220
2221                     }
2222
2223                     if (prob_other!=0) {
2224                         logp+=log10(prob_other);
2225                         log_other+=log10(prob_other);
2226                         log_combine+=log10(prob_other);
2227                         senp+=log10(prob_other);
2228
2229                     }
2230
2231                     if (prob_other!=0) {
2232                         logp+=log10(prob_other);
2233                         log_other+=log10(prob_other);
2234                         log_combine+=log10(prob_other);
2235                         senp+=log10(prob_other);
2236
2237                     }
2238
2239                     if (prob_other!=0) {
2240                         logp+=log10(prob_other);
2241                         log_other+=log10(prob_other);
2242                         log_combine+=log10(prob_other);
2243                         senp+=log10(prob_other);
2244
2245                     }
2246
2247                     if (prob_other!=0) {
2248                         logp+=log10(prob_other);
2249                         log_other+=log10(prob_other);
2250                         log_combine+=log10(prob_other);
2251                         senp+=log10(prob_other);
2252
2253                     }
2254
2255                     if (prob_other!=0) {
2256                         logp+=log10(prob_other);
2257                         log_other+=log10(prob_other);
2258                         log_combine+=log10(prob_other);
2259                         senp+=log10(prob_other);
2260
2261                     }
2262
2263                     if (prob_other!=0) {
2264                         logp+=log10(prob_other);
2265                         log_other+=log10(prob_other);
2266                         log_combine+=log10(prob_other);
2267                         senp+=log10(prob_other);
2268
2269                     }
2270
2271                     if (prob_other!=0) {
2272                         logp+=log10(prob_other);
2273                         log_other+=log10(prob_other);
2274                         log_combine+=log10(prob_other);
2275                         senp+=log10(prob_other);
2276
2277                     }
2278
2279                     if (prob_other!=0) {
2280                         logp+=log10(prob_other);
2281                         log_other+=log10(prob_other);
2282                         log_combine+=log10(prob_other);
2283                         senp+=log10(prob_other);
2284
2285                     }
2286
2287                     if (prob_other!=0) {
2288                         logp+=log10(prob_other);
2289                         log_other+=log10(prob_other);
2290                         log_combine+=log10(prob_other);
2291                         senp+=log10(prob_other);
2292
2293                     }
2294
2295                     if (prob_other!=0) {
2296                         logp+=log10(prob_other);
2297                         log_other+=log10(prob_other);
2298                         log_combine+=log10(prob_other);
2299                         senp+=log10(prob_other);
2300
2301                     }
2302
2303                     if (prob_other!=0) {
2304                         logp+=log10(prob_other);
2305                         log_other+=log10(prob_other);
2306                         log_combine+=log10(prob_other);
2307                         senp+=log10(prob_other);
2308
2309                     }
2310
2311                     if (prob_other!=0) {
2312                         logp+=log10(prob_other);
2313                         log_other+=log10(prob_other);
2314                         log_combine+=log10(prob_other);
2315                         senp+=log10(prob_other);
2316
2317                     }
2318
2319                     if (prob_other!=0) {
2320                         logp+=log10(prob_other);
2321                         log_other+=log10(prob_other);
2322                         log_combine+=log10(prob_other);
2323                         senp+=log10(prob_other);
2324
2325                     }
2326
2327                     if (prob_other!=0) {
2328                         logp+=log10(prob_other);
2329                         log_other+=log10(prob_other);
2330                         log_combine+=log10(prob_other);
2331                         senp+=log10(prob_other);
2332
2333                     }
2334
2335                     if (prob_other!=0) {
2336                         logp+=log10(prob_other);
2337                         log_other+=log10(prob_other);
2338                         log_combine+=log10(prob_other);
2339                         senp+=log10(prob_other);
2340
2341                     }
2342
2343                     if (prob_other!=0) {
2344                         logp+=log10(prob_other);
2345                         log_other+=log10(prob_other);
2346                         log_combine+=log10(prob_other);
2347                         senp+=log10(prob_other);
2348
2349                     }
2350
2351                     if (prob_other!=0) {
2352                         logp+=log10(prob_other);
2353                         log_other+=log10(prob_other);
2354                         log_combine+=log10(prob_other);
2355                         senp+=log10(prob_other);
2356
2357                     }
2358
2359                     if (prob_other!=0) {
2360                         logp+=log10(prob_other);
2361                         log_other+=log10(prob_other);
2362                         log_combine+=log10(prob_other);
2363                         senp+=log10(prob_other);
2364
2365                     }
2366
2367                     if (prob_other!=0) {
2368                         logp+=log10(prob_other);
2369                         log_other+=log10(prob_other);
2370                         log_combine+=log10(prob_other);
2371                         senp+=log10(prob_other);
2372
2373                     }
2374
2375                     if (prob_other!=0) {
2376                         logp+=log10(prob_other);
2377                         log_other+=log10(prob_other);
2378                         log_combine+=log10(prob_other);
2379                         senp+=log10(prob_other);
2380
2381                     }
2382
2383                     if (prob_other!=0) {
2384                         logp+=log10(prob_other);
2385                         log_other+=log10(prob_other);
2386                         log_combine+=log10(prob_other);
2387                         senp+=log10(prob_other);
2388
2389                     }
2390
2391                     if (prob_other!=0) {
2392                         logp+=log10(prob_other);
2393                         log_other+=log10(prob_other);
2394                         log_combine+=log10(prob_other);
2395                         senp+=log10(prob_other);
2396
2397                     }
2398
2399                     if (prob_other!=0) {
2400                         logp+=log10(prob_other);
2401                         log_other+=log10(prob_other);
2402                         log_combine+=log10(prob_other);
2403                         senp+=log10(prob_other);
2404
2405                     }
2406
2407                     if (prob_other!=0) {
2408                         logp+=log10(prob_other);
2409                         log_other+=log10(prob_other);
2410                         log_combine+=log10(prob_other);
2411                         senp+=log10(prob_other);
2412
2413                     }
2414
2415                     if (prob_other!=0) {
2416                         logp+=log10(prob_other);
2417                         log_other+=log10(prob_other);
2418                         log_combine+=log10(prob_other);
2419                         senp+=log10(prob_other);
2420
2421                     }
2422
2423                     if (prob_other!=0) {
2424                         logp+=log10(prob_other);
2425                         log_other+=log10(prob_other);
2426                         log_combine+=log10(prob_other);
2427                         senp+=log10(prob_other);
2428
2429                     }
2430
2431                     if (prob_other!=0) {
2432                         logp+=log10(prob_other);
2433                         log_other+=log10(prob_other);
2434                         log_combine+=log10(prob_other);
2435                         senp+=log10(prob_other);
2436
2437                     }
2438
2439                     if (prob_other!=0) {
2440                         logp+=log10(prob
```

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.

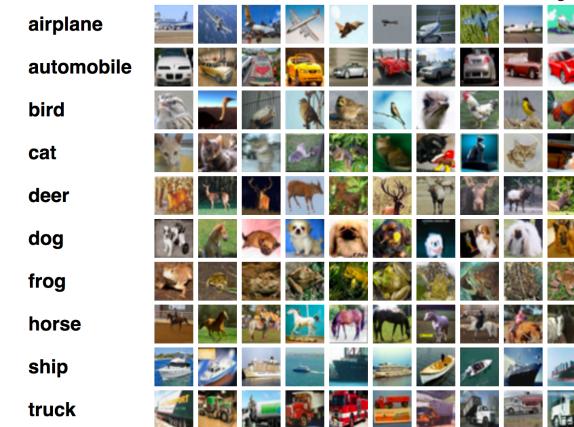
The goal of (supervised) deep learning

- 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].

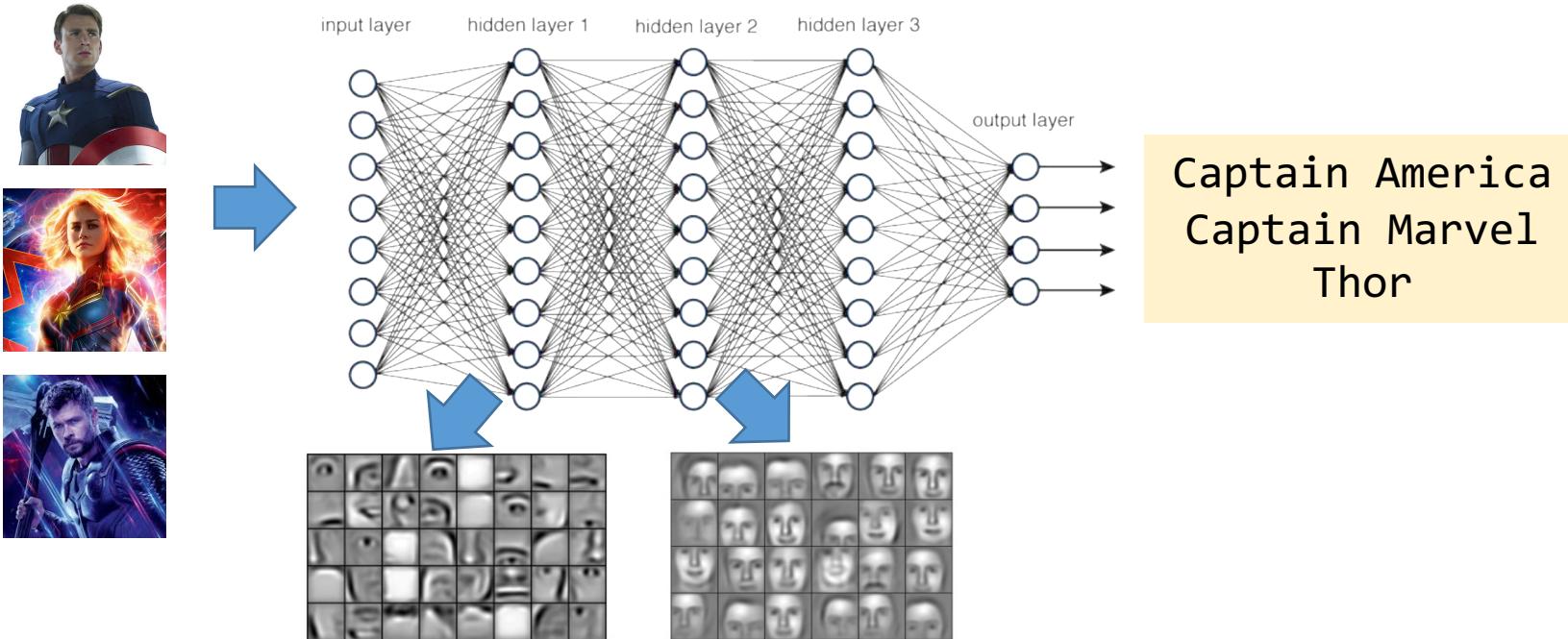
The main weakness of this masterpiece is the lack of practical programming exercises left to a companion web site. But to cover all the practical stuff, the book should have exceeded 775 pages that it already has.

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

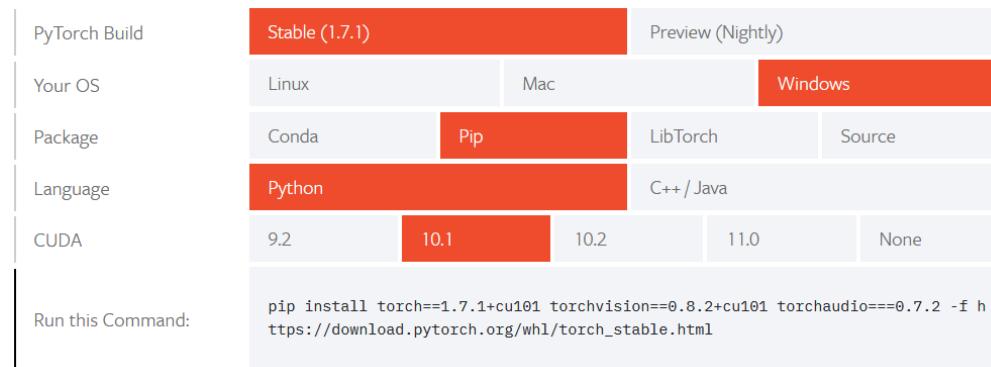
- Installation
 - Libraries for Python
- Data preparation
 - Know how PyTorch stores data / what the data look like
 - Split the data into train/valid/test
- Model preparation
- Model training
 - Define loss function
 - Choose optimization method
 - Train the model on training data
- Model evaluation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

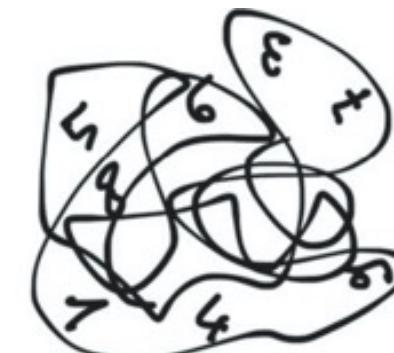
Data structure for representations

- 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 $x[2]=5$ 0D	VECTOR $\begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix}$ 1D	MATRIX $\begin{bmatrix} 4 & 6 & 7 \\ 7 & 3 & 9 \\ 1 & 2 & 5 \end{bmatrix}$ 2D	TENSOR $\begin{bmatrix} 5 & 7 & 1 \\ 9 & 4 & 3 \\ 3 & 5 & 2 \end{bmatrix}$ 3D	TENSOR $x[0, 2, 1] = 2$ $x[1, 3, \dots, 2] = 4$ N-D DATA → N INDICES
--------------------------	---	---	---	---



Tensors are powerful for data representations

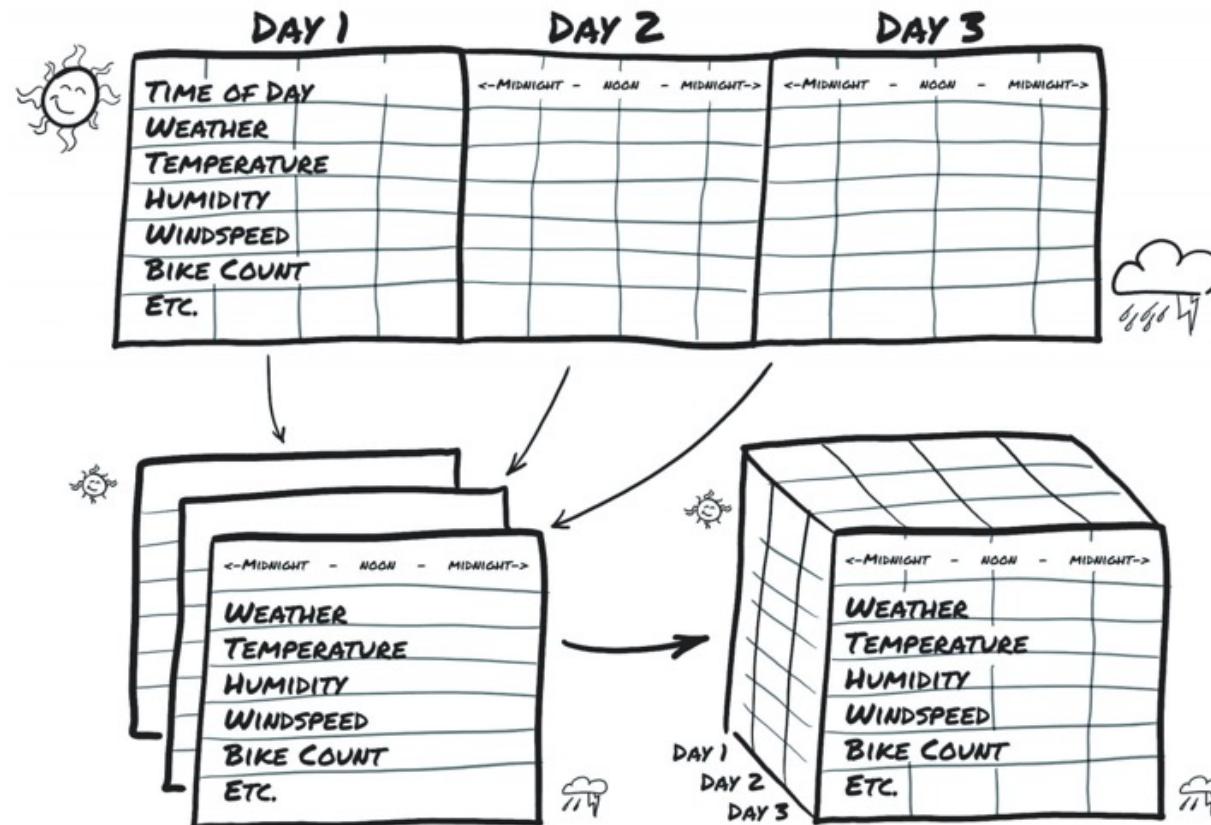
Examples

- 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

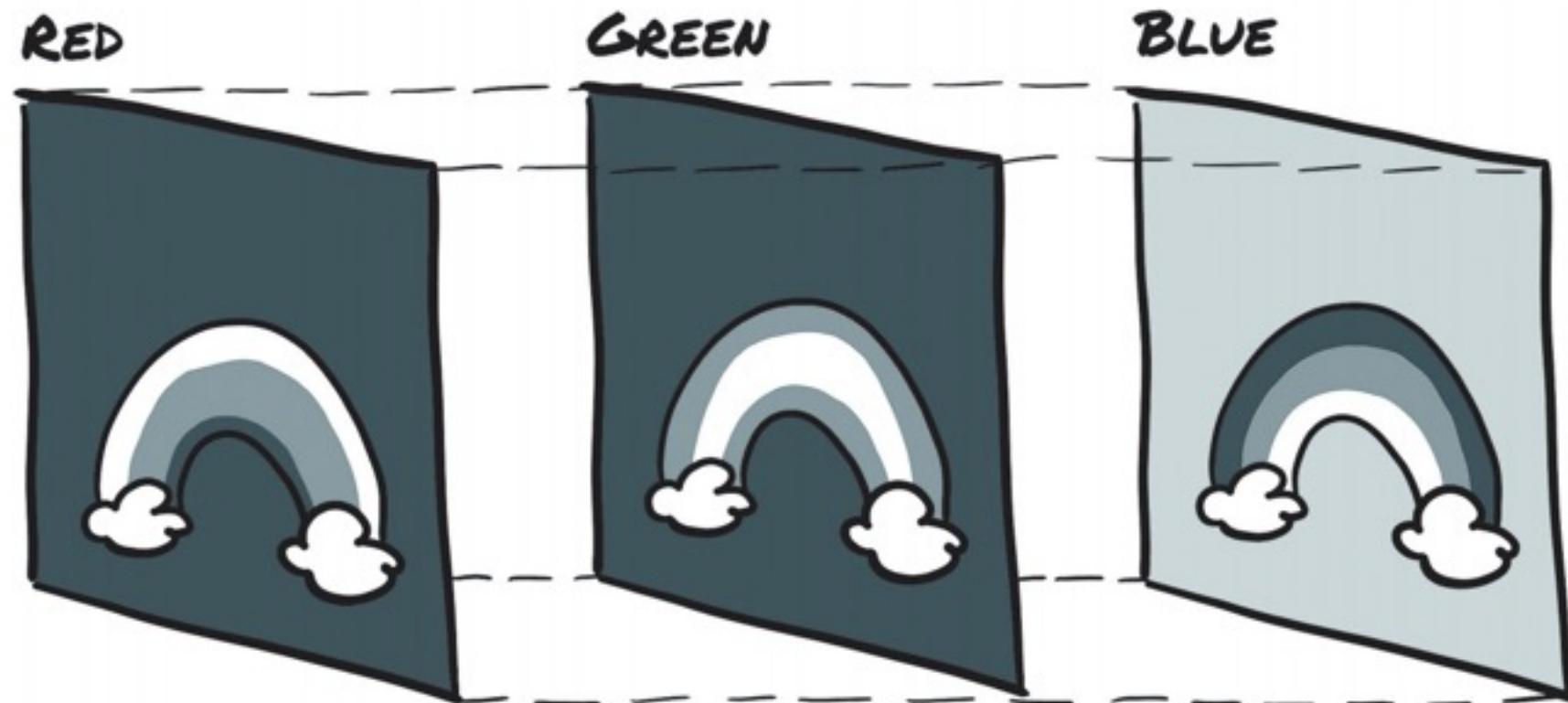
Tensors are powerful for data representations

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



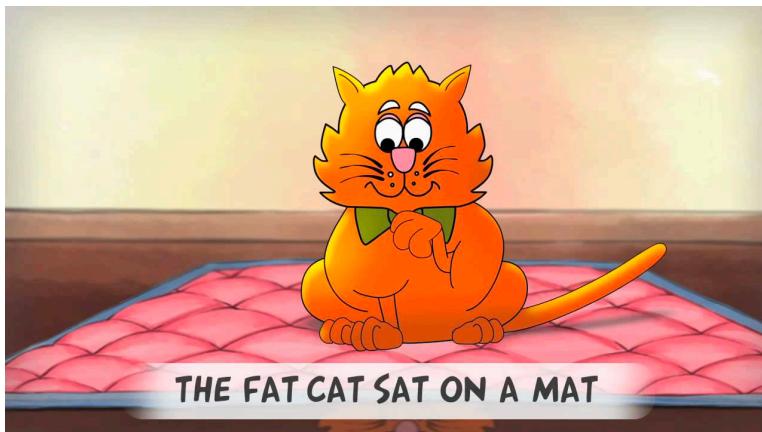
Tensors are powerful for data representations

- 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

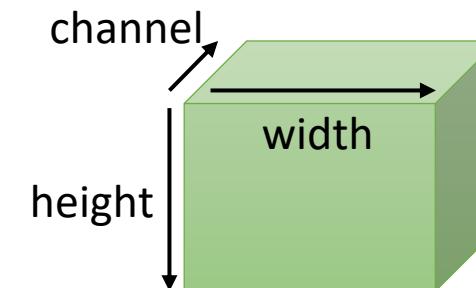
Tensors are powerful for data representations

- 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()
```

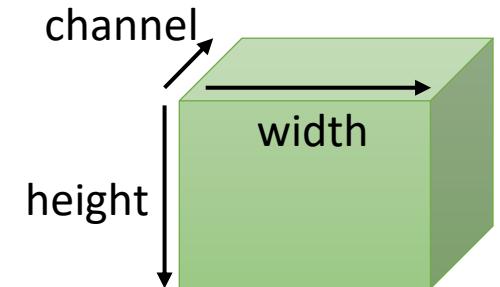
```
torch.float32
```

- 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, :]
```

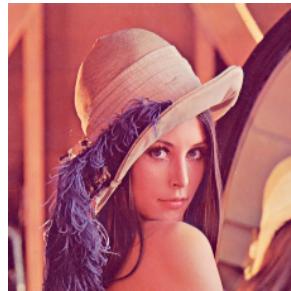
```
tensor([[3., 4.], [5., 6.]])
```

```
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)
```

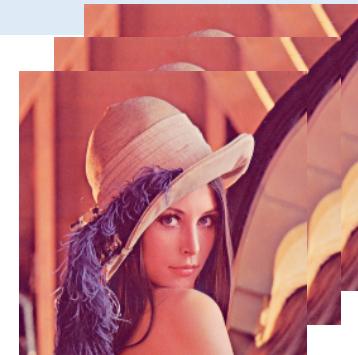
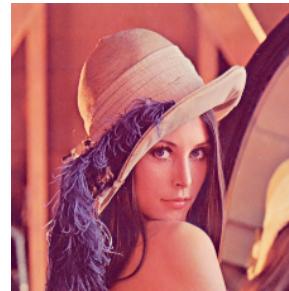
See example in Notebook

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

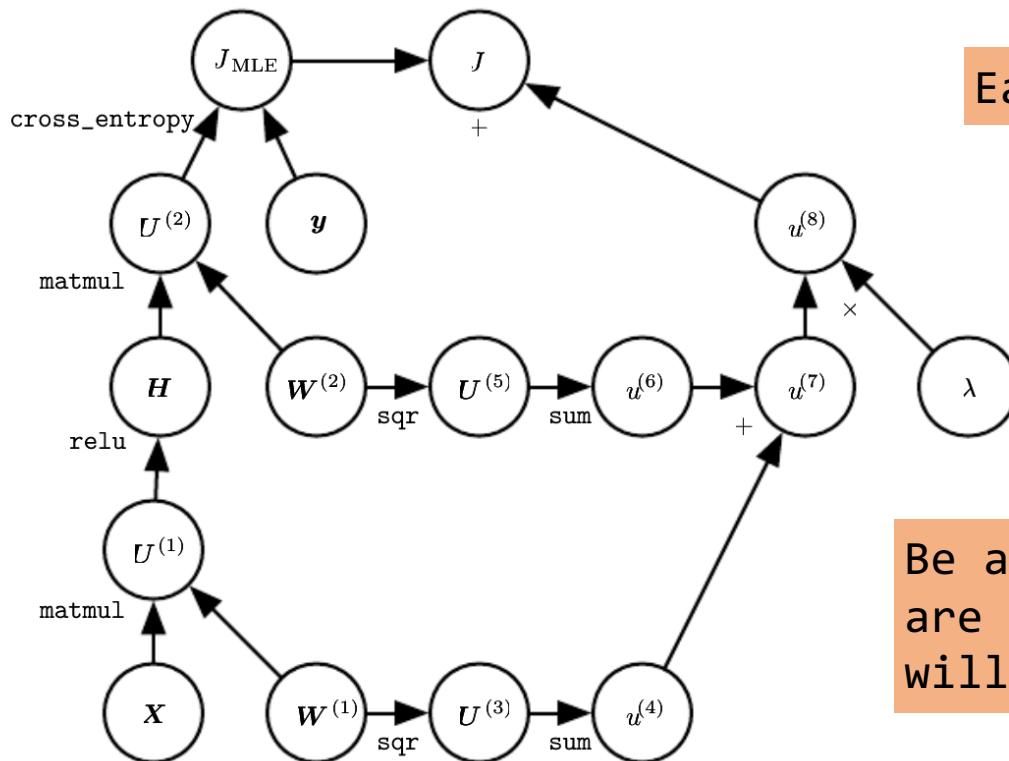
- Compute a scalar to measure the difference between predictions and targets

// Model learning

- Update model parameters

Training through auto-gradient

- Deep Learning book Chapter 6, computation graph example



Each arrow is differentiable!!!

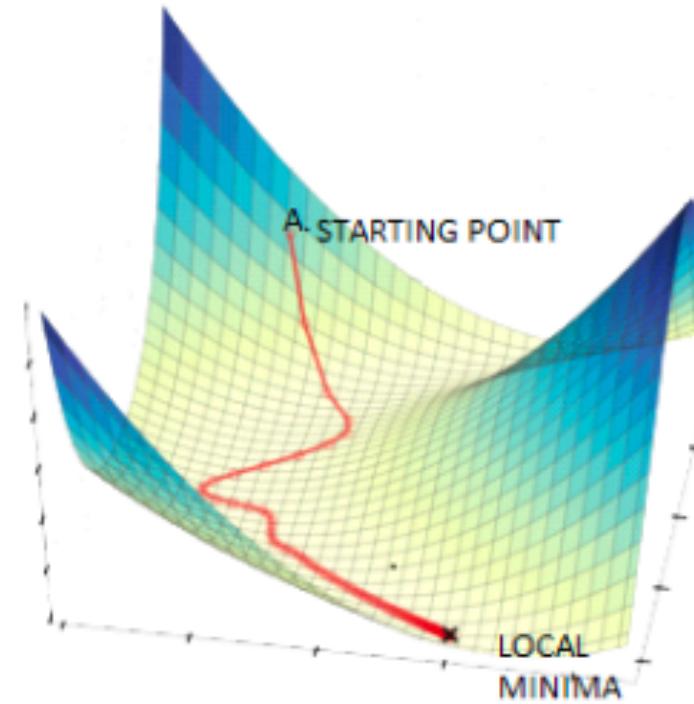
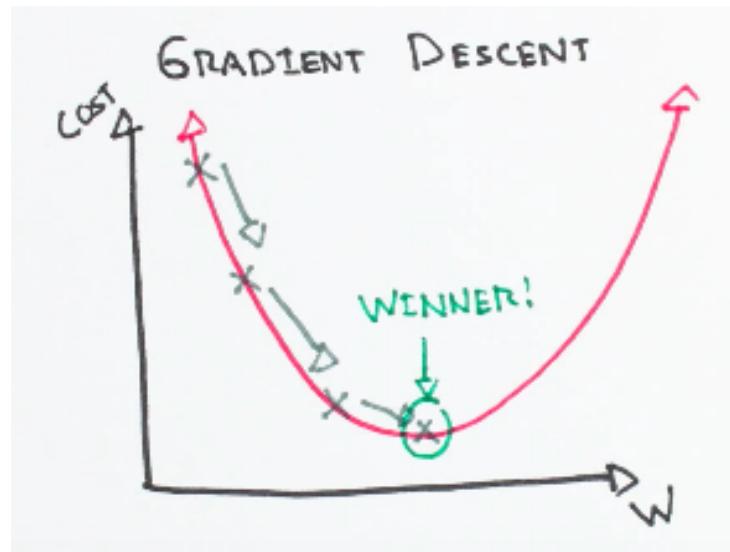
Be aware some functions, e.g. `torch.argmax()`, are not differentiable, and thus no gradient will be propagated past them.

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.

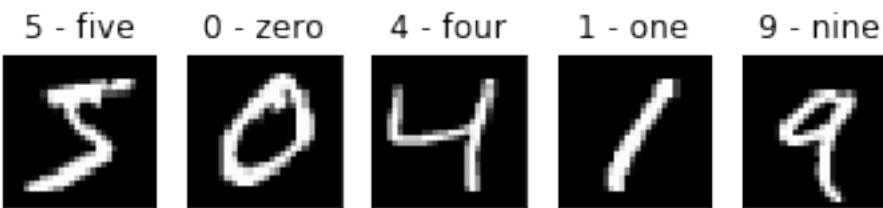


A digit recognition classifier

```
>>> 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

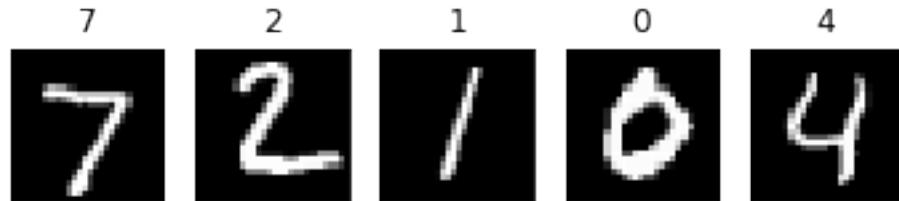
- Then we can train our model.

```
>>> 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

2.3 Customized model

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)

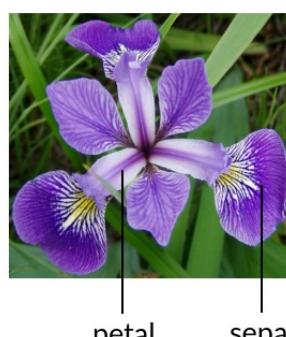
- each belongs to one of the three classes/types of Iris plant

(Setosa, Versicolour, Virginica)

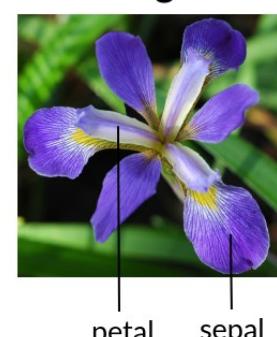
iris setosa



iris versicolor



iris 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

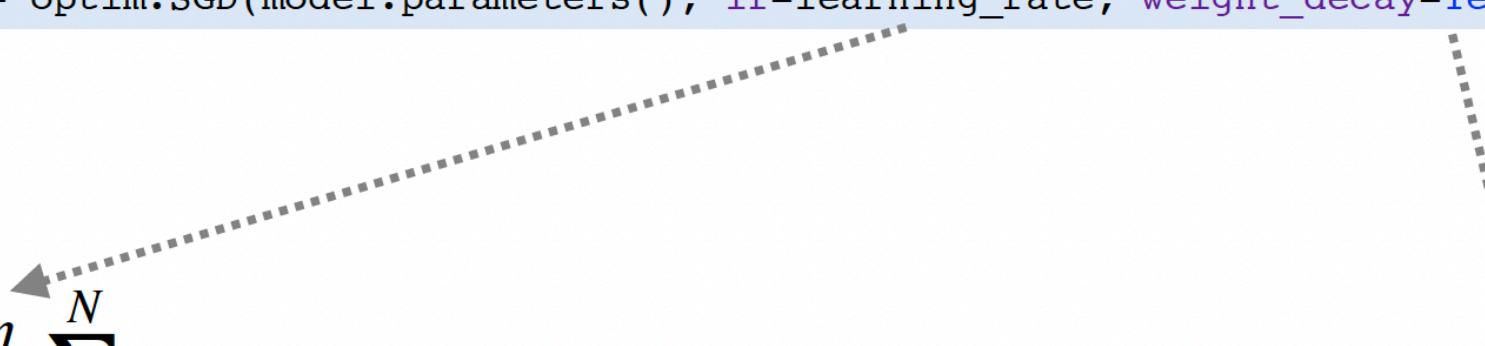
```
>>> ##### Evaluation #####
>>> model.eval()
>>> y_test_pred_output = model(X_test)
>>> _, y_test_pred = torch.max(y_test_pred_output, 1)
>>> acc = torch.true_divide(torch.sum(y_test_pred == y_test), y_test_pred.size()[0])
>>> print('accuracy: {}'.format(acc))
```

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: Explore other optimization methods
- A wide range of optimizers
 - SGD: the classical optimizer
 - RMSprop: a self-adaptive optimizer adaptive gradient
 - 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)
```

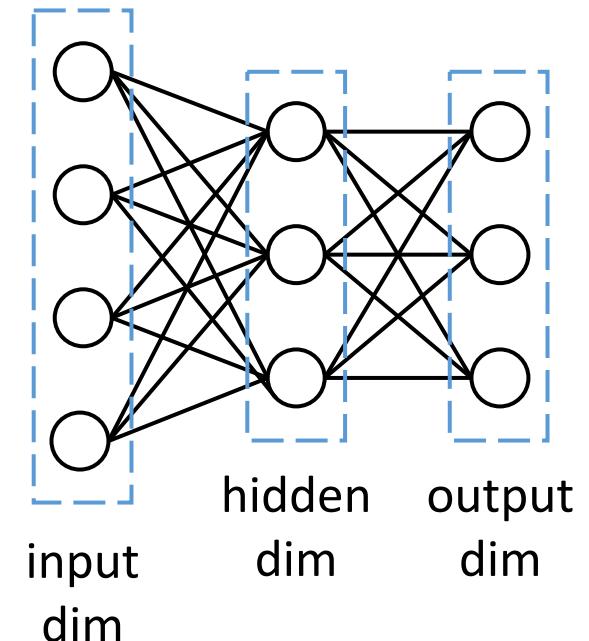
2.3 Customize models

- Basic receipt for customizing a model
 1. Define the parameters
 2. Define the forward function
 3. ~~Define the backward function~~ Pytorch takes care of it :)
- 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__()`

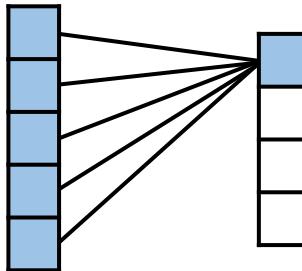
Customize models

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

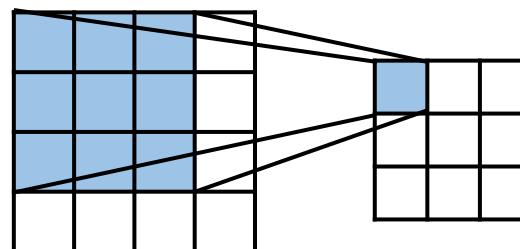
```
>>> class MLP(nn.Module):
>>>     def __init__(self, input_dim, hidden_dim, output_dim):
>>>         super(MLP, self).__init__()
>>>         self.model = nn.Sequential(
>                         nn.Linear(input_dim, hidden_dim),
>                         nn.Linear(hidden_dim, output_dim)
>      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

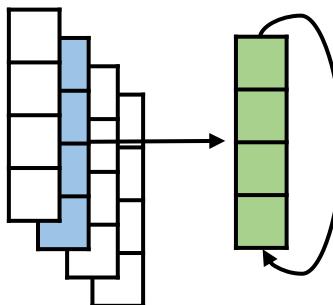


Common building blocks

- Parametric layers

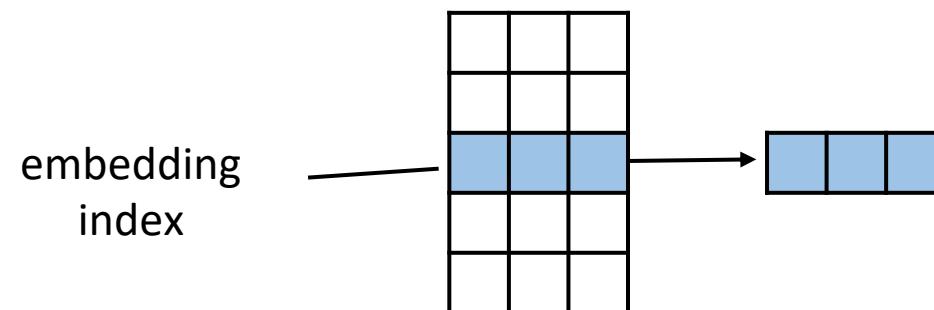
- Recurrent layer (multi-layer)

```
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)`

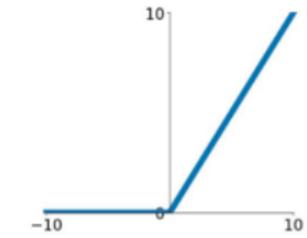
- `F.max_pool2d(kernel_size, stride=None)`

- Dropout layer

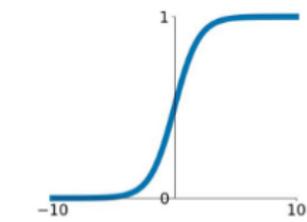
- `nn.Dropout(p=0.5)`

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

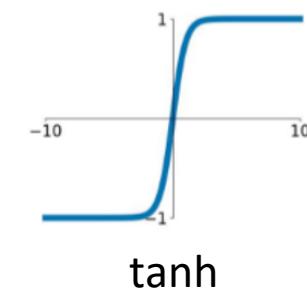
Not recommended!



ReLU



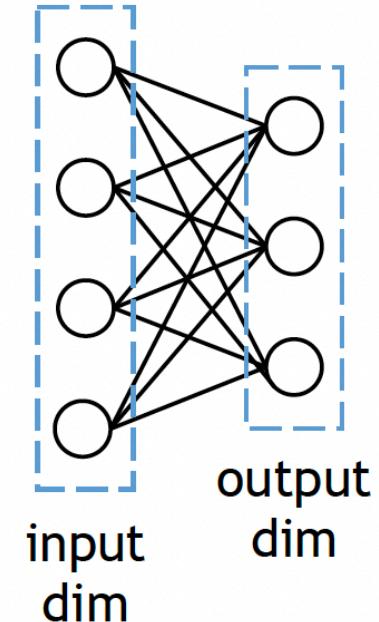
sigmoid



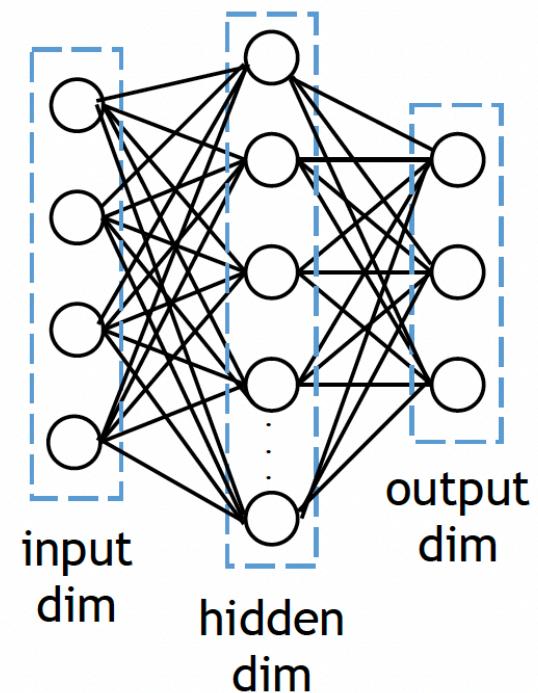
tanh

2.3 Customized model

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

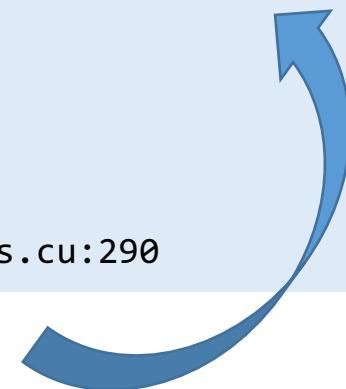
- Figure out where the bug is. A recommended order is
 - Check whether the code can run (-> e.g. shape errors)
 - Check the evaluation code
 - Check the ground truth
 - Check optimizer and learning rate
 - Check model errors
 - Check model capacity

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
```



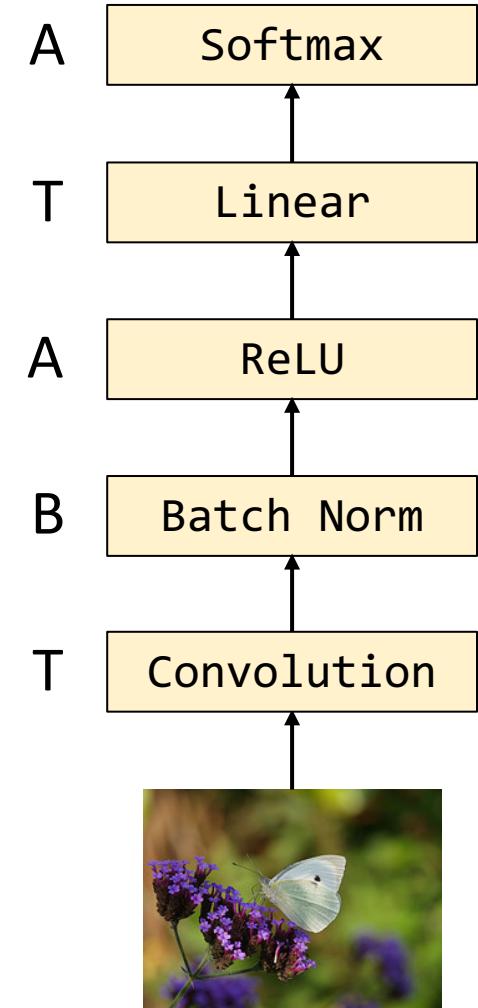
7 / 3 leads to 2 instead of 2.333

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



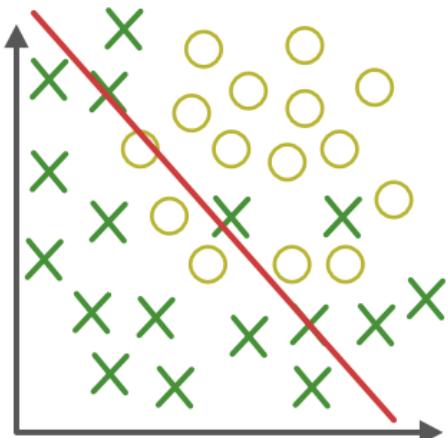
Model errors

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

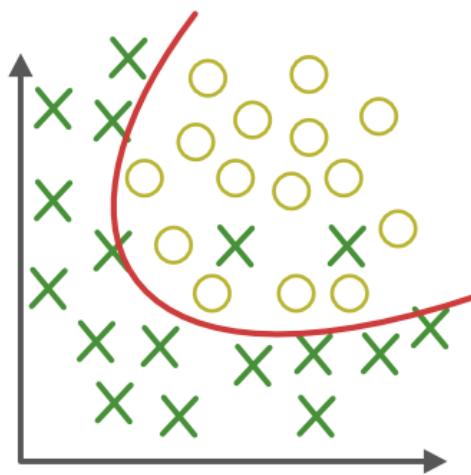


Model capacity

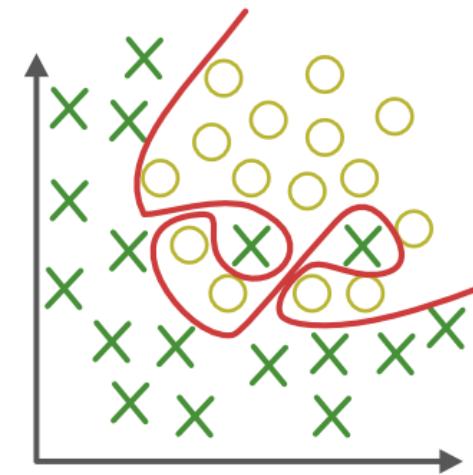
- Capacity: How powerful / complex a model is



Low capacity
Too simple to
explain the
observation



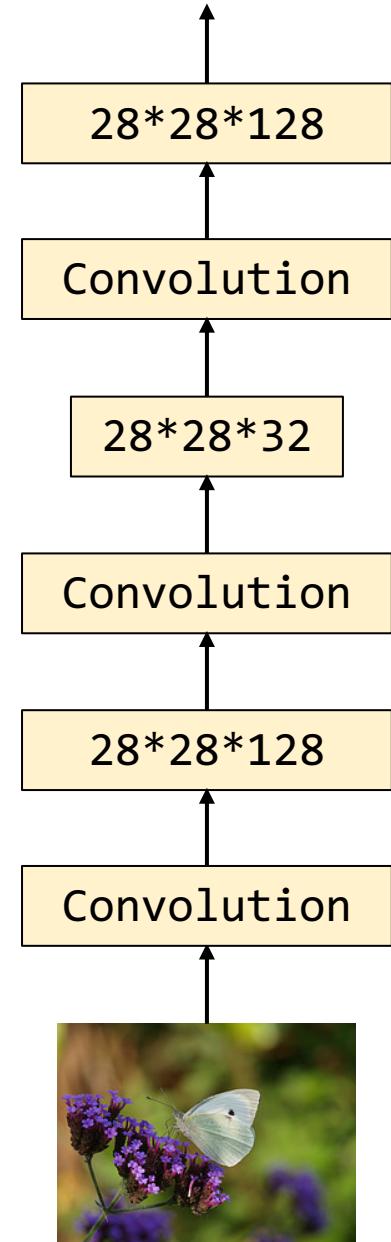
Appropriate capacity



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())
```

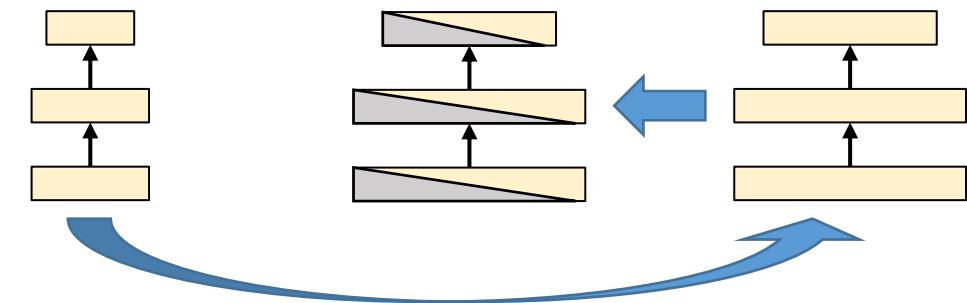
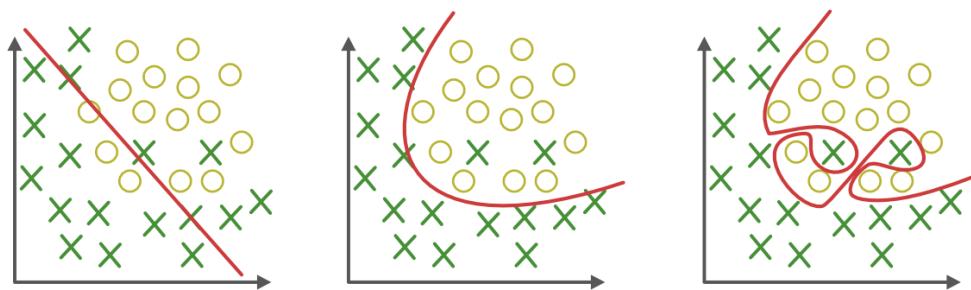
- ResNet18 has ~10M parameters
- GPT-3 has up to 175B parameters.

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!

