Statistical Learning

Lecture 12a - Neural Networks - Deep Learning

ANU - RSFAS

Last Updated: Wed May 25 07:29:37 2022

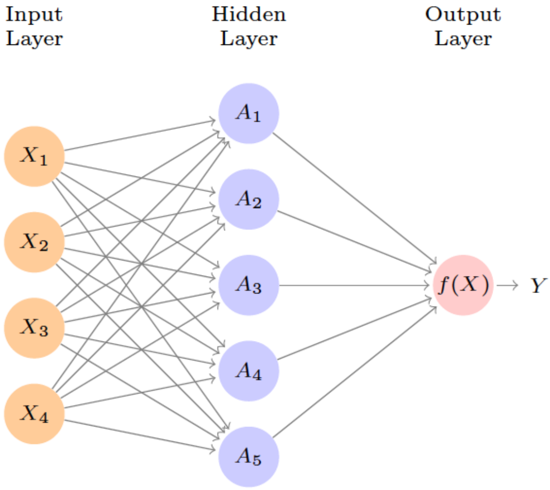
Deep Learning

- Neural networks became popular in the 1980s. Lots of successes and hype.
- Then along came SVMs, Random Forests and Boosting in the 1990s, and Neural Networks took a back seat.
- Re-emerged around 2010 as Deep Learning. By 2020s very dominant and successful. Part of success due to vast improvements in computing power, larger training sets, and software: Tensor flow (Google) and PyTorch (Facebook).

Deep Learning

- Much of the credit goes to three pioneers (computer scientists) and their students: Yann LeCun, Geoffrey Hinton and Yoshua Bengio, who received the 2019 ACM Turing Award for their work in Neural Networks.

Nueral Networks - Single Layer

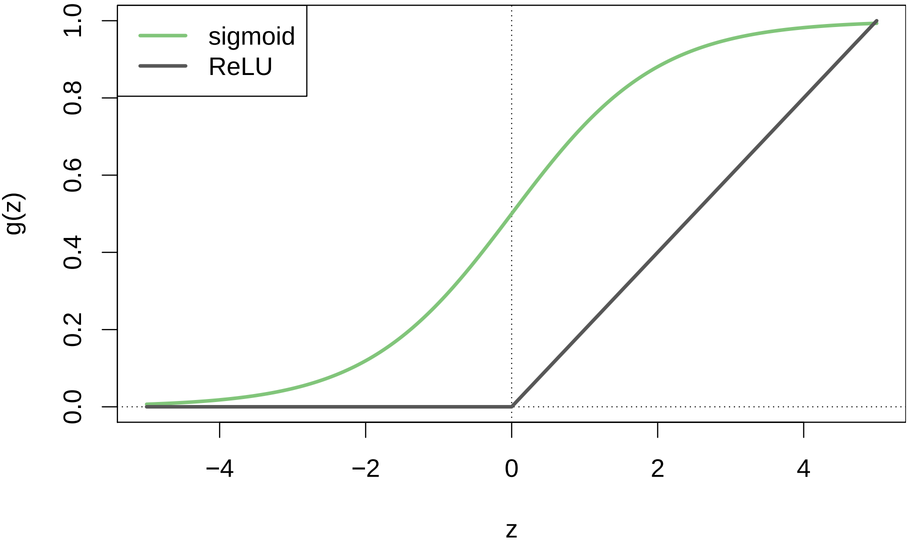


Nueral Networks - Single Layer

$$
\begin{aligned}
f(X) &= \beta_0 + \sum_{i=1}^{K} \beta_k h_k(X) \\
&= \beta_0 + \sum_{i=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j) \\
&= \beta_0 + \sum_{i=1}^{K} \beta_k A_k
\end{aligned}
$$

- $g(\cdot)$ is non-linear **activation function** - specified in advance
- Each A_k is a different transformation $h_k(X)$ of the original features (covariates) - similar in flavor to the basis functions in Chapter 7 for non-linear fits.

- A_k are called the **activations** in the **hidden layer**.
- $g(z)$ is called the activation function. Popular are the **sigmoid** and **rectified linear**
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.
- So the activations are like derived features – nonlinear transformations of linear combinations of the features.
- Moreover, having a nonlinear activation function allows the model to capture complex nonlinearities and interaction effects.


sigmoid
ReLU
g(z)
1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4
z

- Consider a simple example:

$$X = (X_1, X_2), \quad K = 2, \quad p = 2, \quad g(z) = z^2$$

$$\begin{array}{ccc} \beta_0 = 0, & \beta_1 = \frac{1}{4}, & \beta_2 = -\frac{1}{4} \\ w_{10} = 0, & w_{11} = 1, & w_{12} = 1 \\ w_{20} = 0, & w_{21} = 1, & w_{22} = -1 \end{array}$$

- So we have:

$$\begin{aligned} h_1(X) &= (0 + X_1 + X_2)^2 \\ h_2(X) &= (0 + X_1 - X_2)^2 \end{aligned}$$

$$\begin{aligned} f(X) &= 0 + \frac{1}{4}\left((0 + X_1 + X_2)^2\right) - \frac{1}{4}\left((0 + X_1 - X_2)^2\right) \\ &= \frac{1}{4}\left((X_1 + X_2)^2 - (X_1 - X_2)^2\right) \\ &= X_1 X_2 \end{aligned}$$

Fitting

- Fitting a neural network requires estimating the unknown parameters.
- For a quantitative response this is typically done via least-squares:

$$\sum_{i=1}^{n} (y_i - f(x_i))^2$$

- Let's revisit the Baseball Salary data
- This is a regression problem, where the goal is to predict the Salary of a baseball player in 1987 using his performance statistics from 1986.
- After removing players with missing responses, we are left with 263 players and 19 variables.
- We randomly split the data into a training set of 176 players (two thirds), and a test set of 87 players (one third).

```r
library(ISLR2); data("Hitters"); summary(Hitters[,1:8])
```

```
##      AtBat            Hits          HmRun            Runs
##  Min.   : 16.0   Min.   :  1   Min.   : 0.00   Min.   :  0.00
##  1st Qu.:255.2   1st Qu.: 64   1st Qu.: 4.00   1st Qu.: 30.25
##  Median :379.5   Median : 96   Median : 8.00   Median : 48.00
##  Mean   :380.9   Mean   :101   Mean   :10.77   Mean   : 50.91
##  3rd Qu.:512.0   3rd Qu.:137   3rd Qu.:16.00   3rd Qu.: 69.00
##  Max.   :687.0   Max.   :238   Max.   :40.00   Max.   :130.00
##       RBI             Walks            Years            CAtBat
##  Min.   :  0.00   Min.   :  0.00   Min.   : 1.000   Min.   :   19.0
##  1st Qu.: 28.00   1st Qu.: 22.00   1st Qu.: 4.000   1st Qu.:  816.8
##  Median : 44.00   Median : 35.00   Median : 6.000   Median : 1928.0
##  Mean   : 48.03   Mean   : 38.74   Mean   : 7.444   Mean   : 2648.7
##  3rd Qu.: 64.75   3rd Qu.: 53.00   3rd Qu.:11.000   3rd Qu.: 3924.2
##  Max.   :121.00   Max.   :105.00   Max.   :24.000   Max.   :14053.0
```

```r
summary(Hitters[,-c(1:8)])
```

```
##      CHits            CHmRun           CRuns             CRBI        
##  Min.   :   4.0   Min.   :  0.00   Min.   :   1.0   Min.   :   0.00  
##  1st Qu.: 209.0   1st Qu.: 14.00   1st Qu.: 100.2   1st Qu.:  88.75  
##  Median : 508.0   Median : 37.50   Median : 247.0   Median : 220.50  
##  Mean   : 717.6   Mean   : 69.49   Mean   : 358.8   Mean   : 330.12  
##  3rd Qu.:1059.2   3rd Qu.: 90.00   3rd Qu.: 526.2   3rd Qu.: 426.25  
##  Max.   :4256.0   Max.   :548.00   Max.   :2165.0   Max.   :1659.00  
##                                                                      
##      CWalks          League  Division    PutOuts          Assists     
##  Min.   :   0.00   A:175   E:157    Min.   :   0.0   Min.   :  0.0  
##  1st Qu.:  67.25   N:147   W:165    1st Qu.: 109.2   1st Qu.:  7.0  
##  Median : 170.50                    Median : 212.0   Median : 39.5  
##  Mean   : 260.24                    Mean   : 288.9   Mean   :106.9  
##  3rd Qu.: 339.25                    3rd Qu.: 325.0   3rd Qu.:166.0  
##  Max.   :1566.00                    Max.   :1378.0   Max.   :492.0  
##                                                                     
##      Errors          Salary       NewLeague
##  Min.   : 0.00   Min.   :  67.5   A:176    
##  1st Qu.: 3.00   1st Qu.: 190.0   N:146    
##  Median : 6.00   Median : 425.0            
##  Mean   : 8.04   Mean   : 535.9            
##  3rd Qu.:11.00   3rd Qu.: 750.0            
##  Max.   :32.00   Max.   :2460.0            
##                  NA's   :59                
```

```r
hit.na <- na.omit(Hitters)
n <- nrow(hit.na)
set.seed(100)
ntest <- trunc(n / 3)
testid <- sample(1:n, ntest)
```

```r
lfit <- lm(Salary ~ ., data = hit.na[-testid, ])
lpred <- predict(lfit, hit.na[testid, ])
with(hit.na[testid, ], mean(abs(lpred - Salary)))
```

```
## [1] 234.7266
```

```r
library(faraway)
sumary(lfit)
```

```
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 227.612145 114.776318  1.9831 0.049112
## AtBat        -2.108462   0.792265 -2.6613 0.008598
## Hits          7.368520   3.011349  2.4469 0.015519
## HmRun         5.625222   8.024779  0.7010 0.484359
## Runs         -3.015865   4.002006 -0.7536 0.452232
## RBI           0.386283   3.562024  0.1084 0.913782
## Walks         4.574976   2.497751  1.8316 0.068913
## Years        -2.044161  16.833435 -0.1214 0.903503
## CAtBat       -0.426314   0.183867 -2.3186 0.021715
## CHits         1.345224   0.881618  1.5259 0.129070
## CHmRun        2.224998   2.066412  1.0767 0.283257
## CRuns         1.047251   0.968980  1.0808 0.281464
## CRBI         -0.106637   0.940625 -0.1134 0.909885
## CWalks       -0.640045   0.434949 -1.4715 0.143159
## LeagueN      74.793873  96.043111  0.7788 0.437305
## DivisionW   -91.669238  53.891965 -1.7010 0.090939
## PutOuts       0.398458   0.103197  3.8611 0.000165
## Assists       0.302098   0.282269  1.0702 0.286161
## Errors        0.091908   5.456629  0.0168 0.986583
## NewLeagueN  -69.305248  95.146238 -0.7284 0.467456
##
## n = 176, p = 20, Residual SE = 326.96086, R-Squared = 0.59
```

- Let's try with Lasso:

```
x <- scale(model.matrix(Salary ~ . - 1, data = hit.na))
y <- hit.na$Salary
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
cvfit <- cv.glmnet(x[-testid, ], y[-testid],
    type.measure = "mse", alpha=1)
cpred <- predict(cvfit, x[testid, ], s = "lambda.min")
mean(abs(y[testid] - cpred))
```

```
## [1] 231.1828
```

- Let's try the neural network package described in *Extending the Linear Model with R* by Julian Faraway - Chapter 17.

```
library(nnet)
nnmod1 <- nnet(Salary ~ ., size=2, linout=T,
               data = hit.na[-testid, ])
```

```
## # weights:  43
## initial  value 95252155.873999
## final  value 40289843.115943
## converged
```

```
nn.pred <- predict(nnmod1, newdata =
                     hit.na[testid, ])
with(hit.na[testid, ], mean(abs(nn.pred - Salary)))
```

```
## [1] 309.6678
```

```r
bestrss <- nnmod1$value
for(i in 1:100){nnmdl <- nnet(Salary ~ ., size=2, linout=T,
                    data = hit.na[-testid, ], trace=F)

if(nnmdl$value < bestrss){
  bestnn <- nnmdl
  bestrss <- nnmdl$value
  }}
bestnn$value
```

```
## [1] 20470074
```

```
nn.pred <- predict(bestnn, newdata =
                     hit.na[testid, ])
with(hit.na[testid, ], mean(abs(nn.pred - Salary)))
```

```
## [1] 235.2443
```

```
summary(bestnn)
```

```
## a 19-2-1 network with 43 weights
## options were - linear output units
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1
##   -4.55 -282.82   13.68  187.88  401.48  627.12  -92.01  -51.13 -191.79
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1
##  294.39  752.03  346.29 -462.19  -55.52  -50.08   20.09   22.71   82.65
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2
##    0.46   -2.10   -0.36   -0.12    0.38   -0.03    0.25    0.56    0.98
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2
##   -0.58    0.75   -0.70   -0.04    0.31   -0.54   -0.26    0.94   -0.30
##    b->o   h1->o   h2->o
##  114.61  554.34  354.46
```

- $i1 \rightarrow h_1$ is the link between the input variable and the first hidden neuron.
- b refers to the 'bias' (intercept for statisticians) and takes a constant value of 1.

```r
head(nn.pred)
```

```
##                          [,1]
## -Ron Kittle         469.0637
## -Jose Cruz         1023.4016
## -Rance Mulliniks    469.0637
## -Andres Galarraga   114.6055
## -Glenn Wilson       469.0637
## -Argenis Salazar    469.0637
```

Decay: $SSE + \lambda \sum_{i=1}^{2} w_i^2$

```
nnmod2 <- nnet(Salary ~ ., size=25, linout=T,
               data = hit.na[-testid, ], decay=0.2)
```

```
## # weights:  526
## initial  value 95327483.811962
## iter  10 value 34072982.723676
## iter  20 value 27859249.155203
## iter  30 value 27524115.043616
## iter  40 value 27483287.685563
## iter  50 value 26490703.037836
## iter  60 value 26189411.862088
## iter  70 value 23035989.425496
## iter  80 value 22827226.520125
## iter  90 value 22758719.218448
## iter 100 value 22705847.303065
## final  value 22705847.303065
## stopped after 100 iterations
```

```
head(nn.pred)
```

```
##                          [,1]
## -Ron Kittle         766.4460
## -Jose Cruz          766.4460
## -Rance Mulliniks  529.2157
## -Andres Galarraga 650.5051
## -Glenn Wilson       766.4460
## -Argenis Salazar  688.8484
```

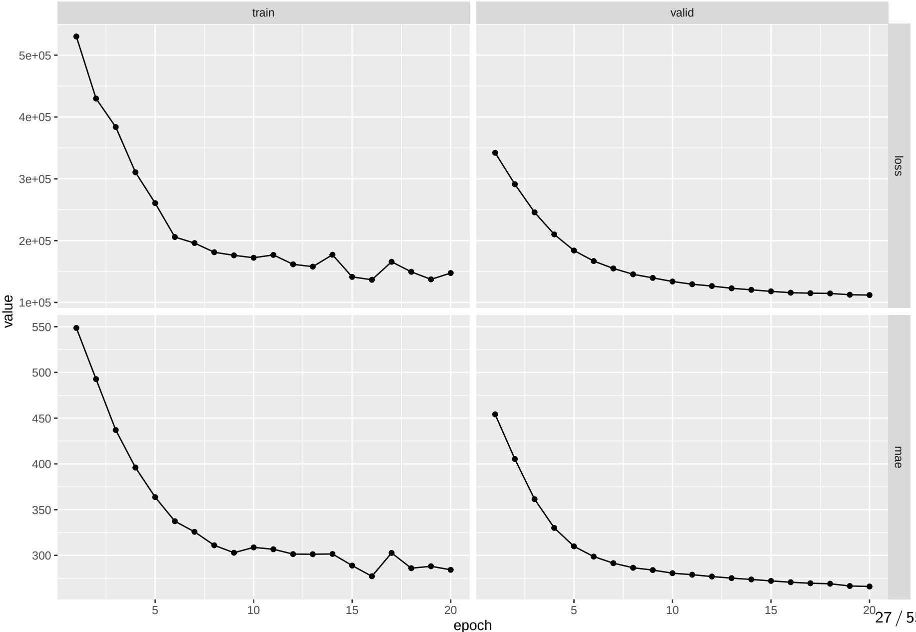
```r
library(torch)
library(luz) # high-level interface for torch
library(torchvision) # for datasets and image transformation
library(torchdatasets) # for datasets we are going to use
library(zeallot)
torch_manual_seed(13)
```

```r
modnn <- nn_module(
  initialize = function(input_size) {
    self$hidden <- nn_linear(input_size, 50)
    self$activation <- nn_relu()
    self$dropout <- nn_dropout(0.4)
    self$output <- nn_linear(50, 1)
  },
  forward = function(x) {
    x %>%
      self$hidden() %>%
      self$activation() %>%
      self$dropout() %>%
      self$output()
  }
)
```

```
modnn <- modnn %>%
  setup(
    loss = nn_mse_loss(),
    optimizer = optim_rmsprop,
    metrics = list(luz_metric_mae())
  ) %>%
  set_hparams(input_size = ncol(x))
```

```r
fitted <- modnn %>%
  fit(
    data = list(x[-testid, ],
                matrix(y[-testid], ncol = 1)),
    valid_data = list(x[testid, ],
                      matrix(y[testid], ncol = 1)),
    epochs = 20
  )
```

```
plot(fitted)
```



train
valid
loss
mae
value
epoch
5e+05
4e+05
3e+05
2e+05
1e+05
550
500
450
400
350
300
5
10
15
20

```r
npred <- predict(fitted, x[testid, ])
mean(abs(y[testid] - npred))
```

```
## torch_tensor
## 355.594
## [ CPUFloatType{} ]
```

Some 'Results' - discussed in the texbook

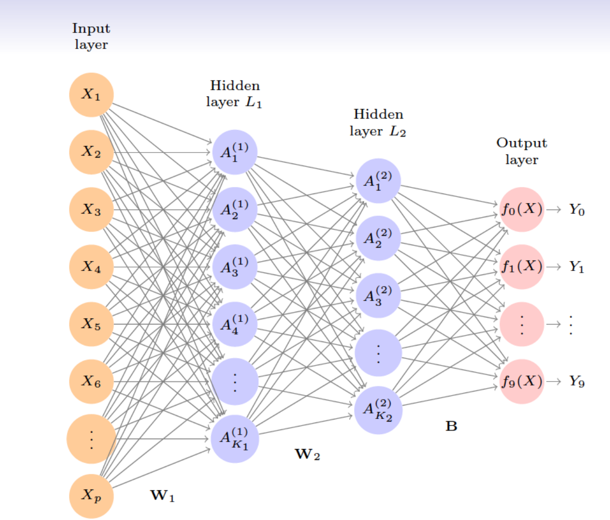
- A linear model was used to fit the training data, and make predictions on the test data. The model has 20 parameters.
- The same linear model was fit with lasso regularization. The tuning parameter was selected by 10-fold cross-validation on the training data. It selected a model with 12 variables having nonzero coefficients.
- A neural network with one hidden layer consisting of 64 ReLU units was fit to the data. This model has 1,409 parameters. **A lot of tweaking involved.**

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1409	257.4	0.54

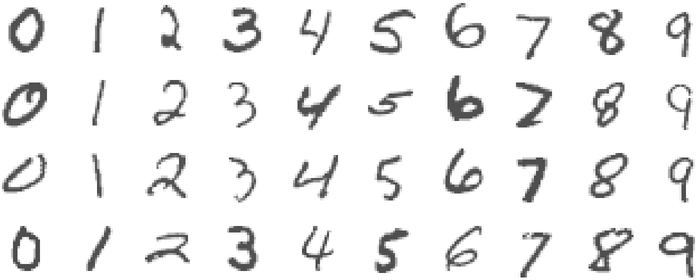
	Coefficient	Std. error	t-statistic	p-value
`Intercept`	-226.67	86.26	-2.63	0.0103
`Hits`	3.06	1.02	3.00	0.0036
`Walks`	0.181	2.04	0.09	0.9294
`CRuns`	0.859	0.12	7.09	< 0.0001
`PutOuts`	0.465	0.13	3.60	0.0005

TABLE 10.3. *Least squares coefficient estimates associated with the regression of* `Salary` *on four variables chosen by lasso on the* `Hitters` *data set. This model achieved the best performance on the test data, with a mean absolute error of 224.8. The results reported here were obtained from a regression on the test data, which was not used in fitting the lasso model.*

Multi-Hidden Layers



Example: MNIST Digits



- Handwritten digits: 28 x 28 grayscale images
- 60K train and 10K test images
- Features (covariates) are the 784 pixel grayscale values $\in (0; 255)$
- Labels (response) are the digit class 0-9

Example: MNIST Digits

- Goal: build a classifier to predict the image class.
- The authors built a two-layer network with 256 units at first layer and 128 units at second layer.
- 10 units at output layer
- Along with intercepts (called biases) there are 235,146 parameters (referred to as weights)

Details of Output Layer

- Let $Z_m = \beta_{0m} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$, $m = 0, 1, 2, \ldots, 9$ be the 10 linear combinations of the activations at the second level.

- The output activation function encodes the **softmax** function (similar to multinomial logistic regression):

$$f_m(X) = P(Y = m|X) = \frac{\exp(Z_m)}{\sum_{\ell=0}^{9} \exp(Z_\ell)}$$

- The model is by minimizing the negative multinomial log-likelihood (or cross-entropy):

$$-\sum_{i=1}^{n} \sum_{m=0}^{9} y_{im} log(f_m(x_i))$$

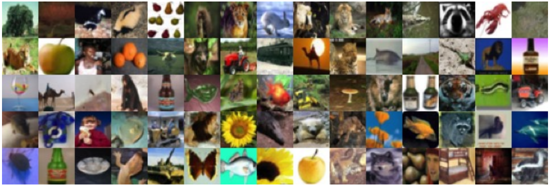
- y_{im} is 1 if true class for observation i is m, else 0 – i.e. **one-hot encoded** (levels of a factor - dummy variables).

Results

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

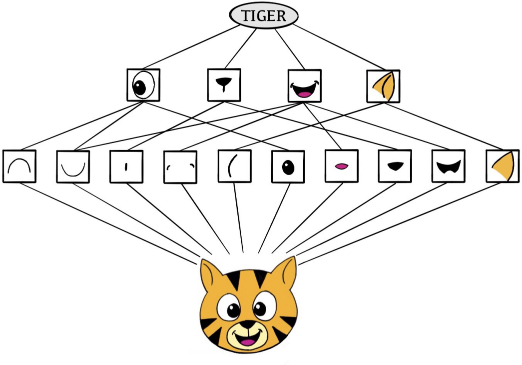
- Early success for neural networks in the 1990s.
- With so many parameters, regularization is essential.
- Very overworked problem – best reported rates are $< 0.5\%$!
- Human error rate is reported to be around 0.2%, or 20 of the 10K test images.

Another Success Story - Image Classification



- This is done through **Convolutional Neural Networks** – CNNs
- Shown are samples from *CIFAR100* database. 32 x 32 color natural images, with 100 classes.
- 50K training images, 10K test images.
- Each image is a three-dimensional array or feature map: 32 x 32 x 3 array of 8-bit numbers. The last dimension represents the three color channels for red, green and blue.

How CNNs Work



- The CNN builds up an image in a hierarchical fashion.
- Edges and shapes are recognized and pieced together to form more complex shapes, eventually assembling the target image.
- This hierarchical construction is achieved using **convolution** and **pooling** layers.

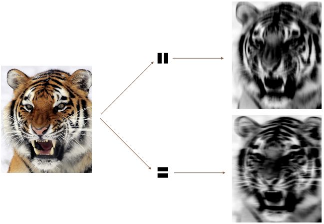
Convolution Filter

$$\text{Input Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \quad \text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

- The filter is itself an image, and represents a small shape, edge etc.
- We slide it around the input image, scoring for matches.
- The scoring is done via dot-products.
- If the subimage of the input image is similar to the filter, the score is high, otherwise low.

Convolution Example



- The idea of convolution with a filter is to find common patterns that occur in different parts of the image.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.
- Since images have three colors channels, the filter does as well: one filter per channel, and dot-products are summed.
- The weights in the filters are **learned** by the network.

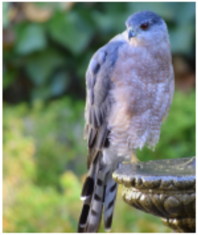
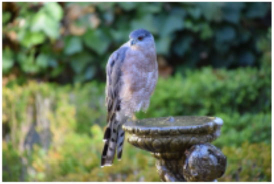
Pooling

$$\text{Max pool} \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

- Each non-overlapping 2 x 2 block is replaced by its maximum.
- This sharpens the feature identification.
- Allows for locational invariance.
- Reduces the dimension by a factor of 4 – i.e. factor of 2 in each dimension.

- Many convolve + pool layers.
- Filters are typically small, e.g. each channel 3 x 3.
- Each filter creates a new channel in convolution layer.
- As pooling reduces size, the number of filters/channels is typically increased.
- Number of layers can be very large. E.g. *resnet50* trained on *imagenet* 1000-class image data base has 50 layers!
- There are many tuning parameters to be selected in constructing such a network, apart from the number, nature, and sizes of each layer.

Using Pretrained CNN Models


```r
img_dir <- "book_images"
image_names <- list.files(img_dir)
num_images <- length(image_names)
x <- torch_empty(num_images, 3, 224, 224)
for (i in 1:num_images) {
   img_path <- file.path(img_dir, image_names[i])
   img <- img_path %>%
     base_loader() %>%
     transform_to_tensor() %>%
     transform_resize(c(224, 224)) %>%
     # normalize with imagenet mean and stds.
     transform_normalize(
       mean = c(0.485, 0.456, 0.406),
       std = c(0.229, 0.224, 0.225)
     )
   x[i,,, ] <- img
}
```

- We then load the trained network. The model has 18 layers, with a fair bit of complexity.

```
model <- torchvision::model_resnet18(pretrained = TRUE)
model$eval() # put the model in evaluation mode
```

- Finally, we classify our six images, and return the top three class choices in terms of predicted probability for each.

```r
preds <- model(x)

mapping <- jsonlite::read_json("https://s3.amazonaws.com/deep-learning-mode
  sapply(function(x) x[[2]])

top3 <- torch_topk(preds, dim = 2, k = 3)

top3_prob <- top3[[1]] %>%
  nnf_softmax(dim = 2) %>%
  torch_unbind() %>%
  lapply(as.numeric)

top3_class <- top3[[2]] %>%
  torch_unbind() %>%
  lapply(function(x) mapping[as.integer(x)])

result <- purrr::map2(top3_prob, top3_class, function(pr, cl) {
  names(pr) <- cl
  pr
})
names(result) <- image_names
print(result)
```

```
## $flamingo.jpg
##    flamingo   spoonbill white_stork
## 0.978211880 0.017045746 0.004742352
##
## $hawk_cropped.jpeg
##      kite       jay    magpie
## 0.6157817 0.2311856 0.1530327
##
## $hawk.jpg
##         eel       agama common_newt
##   0.5391129   0.2527186   0.2081685
##
## $huey.jpg
##          Lhasa Tibetan_terrier        Shih-Tzu
##      0.79760402      0.12013000      0.08226602
##
## $kitty.jpg
##       Saint_Bernard            guinea_pig Bernese_mountain_dog
##           0.3946652             0.3427011            0.2626338
##
## $weaver.jpg
## hummingbird    lorikeet   bee_eater
##   0.3633279   0.3577298   0.2789424
```

Document Classification: IMDB Movie Reviews

- The **IMDB** corpus consists of user-supplied movie ratings for a large collection of movies. Each has been labeled for sentiment as **positive** or **negative**.

- Labeled training and test sets, each consisting of 25,000 reviews, and each balanced with regard to sentiment.

- We wish to build a classifier to predict the sentiment of a review.

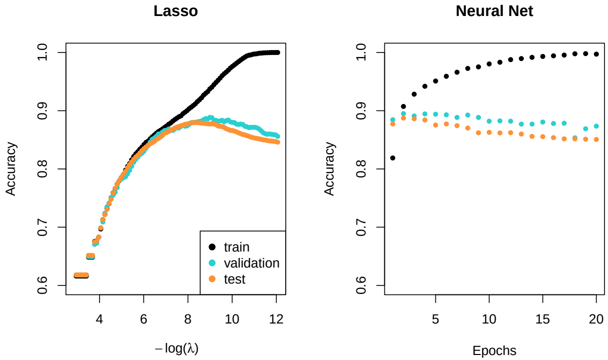
Featurization (Derived Covariates): Bag-of-Words

- Documents have different lengths, and consist of sequences of words.
- How do we create features X to characterize a document?

- From a dictionary, identify the 10K most frequently occurring words.
- Create a binary vector of length $p = 10\text{K}$ for each document, and score a 1 in every position that the corresponding word occurred.
- With n documents, we now have a $n \times p$ sparse feature matrix X.
- We compare a lasso logistic regression model to a two-hidden-layer neural network on the next slide. (No convolutions here!)

- *Bag-of-words* are **unigrams**. We can instead use **bigrams** (occurrences of adjacent word pairs), and in general **m-grams**.

Results



- Simpler lasso logistic regression model works as well as neural network in this case.
- **glmnet** was used to fit the lasso model, and is very effective because it can exploit sparsity in the X matrix.

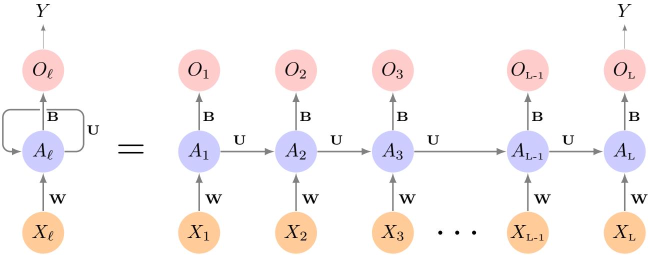
Recurrent Neural Networks

Often data arise as sequences:

- Documents are sequences of words, and their relative positions have meaning.
- Time-series such as weather data or financial indices.
- Recorded speech or music.
- Handwriting, such as doctor's notes.

RNNs build models that take into account this sequential nature of the data, and build a memory of the past.

Recurrent Neural Networks



- For the IMBD data - each document represented as a series of L words (X_ℓ) represents a word
- An application of this applied to IMBD data described in the textbook only achieved a achieved 76% accuracy.
- The lasso performed still better – however expanded versions of RNNs have now performed better.

When to Use Deep Learning

- CNNs have had enormous successes in image classification and modeling, and are starting to be used in medical diagnosis. Examples include digital mammography, ophthalmology, MRI scans, and digital X-rays.
- RNNs have had big wins in speech modeling, language translation, and forecasting.

Should we always use deep learning models?

- Often the big successes occur when the **signal-to-noise ratio** is high – e.g. image recognition and language translation. Datasets are large, and overfitting is not a big problem.
- For noisier data, simpler models can often work better.

Final Graph of the Course