MONASH University

ETC3555

Statistical Machine Learning

Deep Neural Networks

4 September 2018

Deep learning in R with Keras

```r
library(tidyverse)
library(keras)

keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = "softmax") %>%
  compile(loss = "categorical_crossentropy",
          optimizer = optimizer_rmsprop(),
          metrics = c("accuracy")) %>%
  fit(x_train, y_train, epochs = n_epochs, batch_size = 128, validation_split = 0.2)
```

Some design choices

- Network architecture
- Activation functions for hidden and output layers
- Weight initialization
- Optimization algorithm
- Regularization method
- Hyperparameter selection procedure
- ...

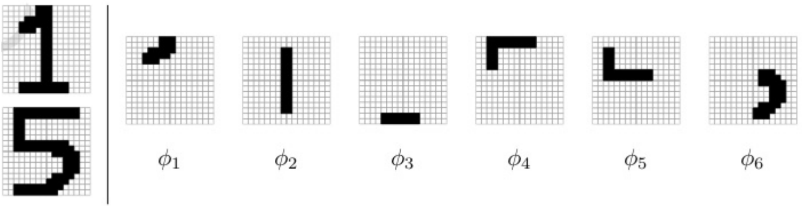
Deep neural networks

- The universal approximation theorem states that a single hidden layer with *enough hidden units* can approximate *any* target function with any desired accuracy.
- In other words, regardless of what the target function is, a large network will be able to *represent* this function.

Deep neural networks

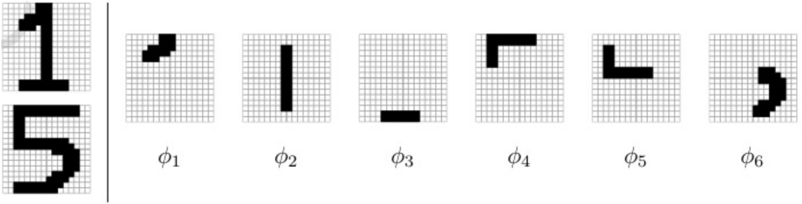
1. We are not guaranteed that the training algorithm will be able to *learn* that function. The algorithm can fail due to overfitting and the hard optimization problem.
2. That may not be a natural way to represent the target function. Many layers (i.e. a composition of several simpler functions) more closely mimics human learning.
3. Empirically, a deeper network does seem to result in better generalization for a wide variety of tasks. This is an area of active research.

Example



We consider the digit recognition problem to classify 1 versus 5. How to classify '1' and '5' from these basic shapes/features?

Example



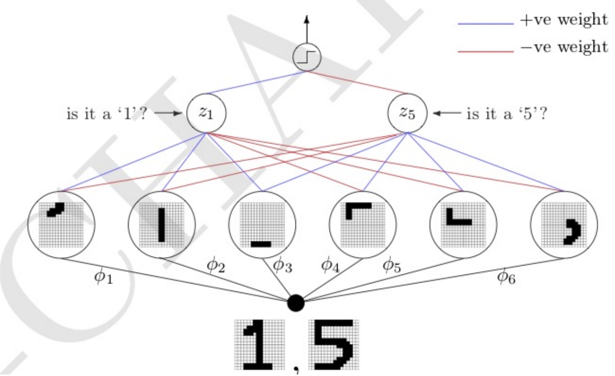
We consider the digit recognition problem to classify 1 versus 5. How to classify '1' and '5' from these basic shapes/features?

- '1' should contain a ϕ_1, ϕ_2 and ϕ_3
- '5' should contain a ϕ_3, ϕ_4, ϕ_5, ϕ_6 and perhaps a little ϕ_1

We want the value of these features to be large (close to 1) if its corresponding feature is in the input image and small (close to -1) if not.

Example

- We build complex Boolean functions from the 'basic' functions AND and OR. We want to mimic that process here. We assume we have feature functions ϕ_i which compute the presence (+1) or absence (-1) of the corresponding feature.



Exercise

Exercise 7.18

Since the input $\mathbf{x}$ is an image it is convenient to represent it as a matrix $[x_{ij}]$ of its pixels which are black ($x_{ij} = 1$) or white ($x_{ij} = 0$). The basic shape ϕ_k identifies a set of these pixels which are black.

(a) Show that feature ϕ_k can be computed by the neural network node

$$\phi_k(\mathbf{x}) = \tanh\left(w_0 + \sum_{ij} w_{ij} x_{ij}\right).$$

(b) What are the inputs to the neural network node?

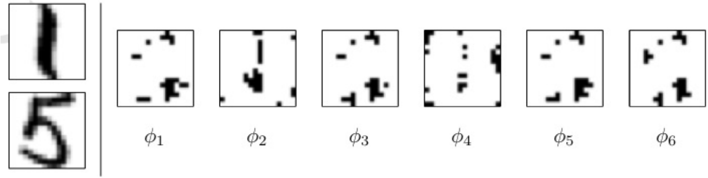
(c) What do you choose as values for the weights? *[Hint: consider separately the weights of the pixels for those $x_{ij} \in \phi_k$ and those $x_{ij} \notin \phi_k$.]*

(d) How would you choose w_0? (Not all digits are written identically, and so a basic shape may not always be exactly represented in the image.)

(e) Draw the final network, filling in as many details as you can.

Example

- A deep network architecture with $[d^{(0)}, d^{(1)}, d^{(2)}, d^{(3)}] = [256, 6, 2, 1]$
- The following figure shows what the 6 hidden units in the first hidden layer **learned**. The pixels corresponding to the top 20 incoming weights are shown.



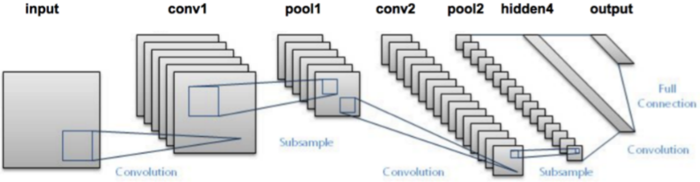
Deep Network Architecture	E_{in}	E_{test}
$[256, 3, 2, 1]$	0	0.170%
$[256, 6, 2, 1]$	0	0.187%
$[256, 12, 2, 1]$	0	0.187%
$[256, 24, 2, 1]$	0	0.183%

Example

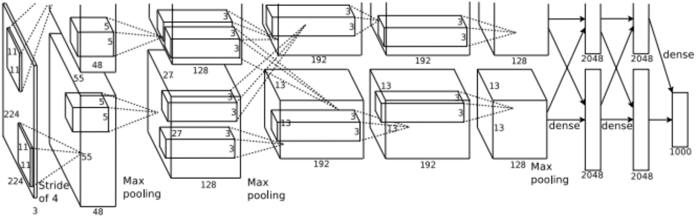
- Our ’deep network’ has just 2 hidden layers. In a more complex problem, there would be a hierarchy of “basic” features.
- “The first layer constructs a **low-level** representation of basic shapes”
- “The next layer builds a **higher level** representation from these basic shapes”
- We obtain more complex representations in terms of simple parts from previous layers. An ’intelligent’ decomposition of the problem.

Deep architectures

LeNet

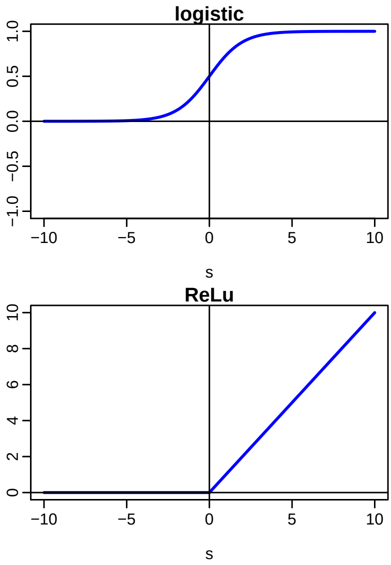


AlexNet



Many others. Source: `https://bit.ly/2mWCUw1`

Activation functions



Rectified linear unit (ReLu)

- $\theta(s) = max(0, s)$
- The most popular activation function in modern deep learning
- Better gradient propagation compared to sigmoidal activation functions that saturate in both directions (less vanishing gradient problems)
- Sigmoidal activation functions have an almost zero gradient at certain regions. This is an undesirable property since it results in slow learning.
- Efficient computation
- In a randomly initialized network, only about half of the hidden units are activated (having a non-zero output)

Rectified linear unit (ReLu)

- Potential problems: Non-differentiable at zero, unbounded, etc.
- Dying ReLU problem: if a ReLu unit become inactive for all inputs, no gradients flow backward through that unit, and it 'dies'
- Leaky ReLU: $\theta(s) = \begin{cases} s, & \text{if } s \geq 0 \\ as & \text{otherwise} \end{cases}$ where $a > 0$ is a small positive number
- Other variants

Softmax (output) activation function

In logistic regression, we compute

$$P(y|\boldsymbol{x}) = \begin{cases} h(\boldsymbol{x}) = \theta(w^T\boldsymbol{x}) = \theta(s) & \text{for } y = +1; \\ 1 - h(\boldsymbol{x}) & \text{for } y = -1, \end{cases}$$

where $\theta(\cdot)$ is the logistic function.

The *softmax* function is a generalized logistic activation function used for multiclass classification. It has both multivariate input and output.

For a classification problem with K classes, we compute

$$P(y = j|\boldsymbol{x}) = \theta(\boldsymbol{s})_j = \theta(\boldsymbol{s})_j = \frac{e^{s_j}}{\sum_{k=1}^{K} e^{s_k}},$$

where $\boldsymbol{s} = (s_1, s_2, \ldots, s_K)^T$, $s_j = \boldsymbol{w}_j^T\boldsymbol{x}$ and $j = 1, 2, \ldots, K$.

Softmax (output) activation function

- Softmax vs "hardmax "
  - Softmax: $\boldsymbol{s} = (5, 2, -1, 3)^T \xrightarrow{\theta} (0.84, 0.04, 0, 0.12)^T$
  - Hardmax: $\boldsymbol{s} = (5, 2, -1, 3)^T \xrightarrow{\theta} (1, 0, 0, 0)^T$

$$\theta(\boldsymbol{s})_j = \begin{cases} 1 & \text{if } j = \text{argmax}(s_1, s_2, \ldots, s_K); \\ 0 & \text{otherwise.} \end{cases}$$

- The softmax behaves as a smooth function (which can be differentiated) and approximates the indicator function.
- The outputs are represented using binary vectors and the multi-class cross-entropy (or logloss) is used as loss function

Regularization

- L_1 and L_2 norm regularization (**weight decay**, **weight elimination**, etc)
- **Early stopping**
- Dropout
- Data augmentation
- ...

L_2 regularization: weight decay

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\sum_{\ell,i,j}(w_{ij}^{(\ell)})^2$$

L_2 regularization: weight decay

$$E_{\text{aug}}(\mathbf{w}) \quad = \quad E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum_{\ell,i,j} (w_{ij}^{(\ell)})^2$$

What is $\frac{\partial E_{\text{aug}}}{\partial w_{ij}^{(l)}}$?

L_2 regularization: weight decay

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\sum_{\ell,i,j}(w_{ij}^{(\ell)})^2$$

What is $\dfrac{\partial E_{\text{aug}}}{\partial w_{ij}^{(l)}}$?

$$\frac{\partial E_{\text{aug}}(\mathbf{w})}{\partial \mathrm{W}^{(\ell)}} = \frac{\partial E_{\text{in}}(\mathbf{w})}{\partial \mathrm{W}^{(\ell)}} + \frac{2\lambda}{N}\mathrm{W}^{(\ell)}$$

- A component in the negative direction of $\boldsymbol{w}$ is added to the weight update.
- We can compute the first term using backpropagation

L_2 regularization: weight elimination

$$E_{\text{aug}}(\mathbf{w}, \lambda) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum_{\ell,i,j} \frac{(w_{ij}^{(\ell)})^2}{1 + (w_{ij}^{(\ell)})^2}$$

L_2 regularization: weight elimination

$$E_{\text{aug}}(\mathbf{w}, \lambda) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum_{\ell,i,j} \frac{(w_{ij}^{(\ell)})^2}{1 + (w_{ij}^{(\ell)})^2}$$

What is $\frac{\partial E_{\text{aug}}}{\partial w_{ij}^{(l)}}$?

L_2 regularization: weight elimination

$$E_{\text{aug}}(\mathbf{w}, \lambda) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N} \sum_{\ell,i,j} \frac{(w_{ij}^{(\ell)})^2}{1 + (w_{ij}^{(\ell)})^2}$$

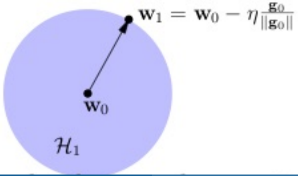
What is $\frac{\partial E_{\text{aug}}}{\partial w_{ij}^{(l)}}$?

$$\frac{\partial E_{\text{aug}}}{\partial w_{ij}^{(\ell)}} = \frac{\partial E_{\text{in}}}{\partial w_{ij}^{(\ell)}} + \frac{2\lambda}{N} \cdot \frac{w_{ij}^{(\ell)}}{(1 + (w_{ij}^{(\ell)})^2)^2}$$

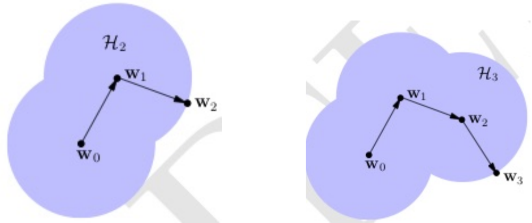
Many other weight based complexity penalties ...

Early stopping

- With more gradient descent iterations, more of our hypothesis set is explored. With fewer iterations, we explore a smaller hypothesis set + better generalization
- With fixed-step gradient descent, we do:
  1. We start at weights $\boldsymbol{w}_0$
  2. We take a step size η to $\boldsymbol{w}_1 = \boldsymbol{w}_0 - \eta \frac{\boldsymbol{g}_0}{\|\boldsymbol{g}_0\|}$
- We implicitly searched the following hypothesis set: $\mathcal{H}_1 = \{\boldsymbol{w} : \|\boldsymbol{w} - \boldsymbol{w}_0\| \leq \eta\}$ and picked the hypothesis $\boldsymbol{w}_1$ with minimum E_{in}.



Early stopping



- $\mathcal{H}_2 = \mathcal{H}_1 \cup \{\boldsymbol{w} : \|\boldsymbol{w} - \boldsymbol{w}_1\| \leq \eta\}$,
  $\mathcal{H}_3 = \mathcal{H}_2 \cup \{\boldsymbol{w} : \|\boldsymbol{w} - \boldsymbol{w}_2\| \leq \eta\}$, and $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \subset \dots$
- As t increases, $E_{\text{in}}(\boldsymbol{w}_t)$ decreases but the complexity of $\mathcal{H}_t$ increases
- It may be better to stop early at some t^*, well before reaching a minimum of E_{in}. A validation set can be used to monitor validation error and determine when to stop training.