CSCI E-89B Introduction to Natural Language Processing

Harvard Extension School

Dmitry Kurochkin

Fall 2024
Lecture 1

Contents

Contents

Support Teaching Staff Members

TBA

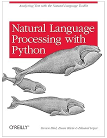
TBA

TBA

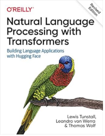
Contents

Textbooks

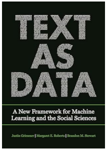
Steven Bird,
Natural Language Processing with Python,
O'Reilly Media, Inc., 2016
ISBN: 978-1-491-91342-0
Electronic copy can be obtained via Harvard Library.



Lewis Tunstall, Leandro von Werra, Thomas Wolf,
Natural Language Processing with Transformers,
O'Reilly Media, Inc., 2022
ISBN: 978-1-098-13676-5
Electronic copy can be obtained via Harvard Library.



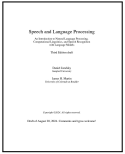
Justin Grimmer, Margaret E. Roberts, Brandon M. Stewart, *Text as Data*, Chapter 13 (Topic Models).
Princeton University Press, 2022
ISBN: 978-0-691-20755-1



Alternative: Roberts, M., Stewart, B., & Tingley, D. (2019). stm: R Package for Structural Topic Models. Journal of Statistical Software, 91(2), 1-40.

Optional Textbook

Daniel Jurafsky, James H. Martin,
Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,
O'Reilly Media, Inc., 2023
ISBN: 979-1-221-47684-2


Speech and Language Processing

An Introduction to Natural Language Processing,
Computational Linguistics, and Speech Recognition
with Language Models

Third Edition draft

Daniel Jurafsky
Stanford University

James H. Martin
University of Colorado at Boulder

Copyright ©2024. All rights reserved.

Draft of August 20, 2024. Comments and typos welcome!

Electronic copy of the book is available via the author's webpage: [Speech and Language Processing.]

Prerequisites

Calculus

Basic knowledge of calculus:

- Derivatives
- Chain Rule
- Partial derivatives
- Chain Rule
- Gradient etc.

Prerequisites

Probability and Statistics

Basic understanding of probability and statistics:

- Random variables
- Probability distributions
- Expectations etc.

Prerequisites

Python

Python programming equivalent to CSCI E-7:

- Data types
- For loops
- If. . . elif . . . else
- Functions
- Classes/Objects etc.

Grading

Assignments, Quizzes, Exams

$$\begin{aligned} \text{Grade} = & \; 0.65 \cdot \text{Homework (weekly, starting Septemebr 22)} \\ & + 0.20 \cdot \text{Quizzes (two per week, starting September 16)} \\ & \; 0.15 \cdot \text{Final Project (due December 16, 11:59 pm ET)} \end{aligned}$$

Homework Assignments

- Unless otherwise specified, homework assignments are due every Sunday by 11:59 PM (Eastern Time).
- Solutions to the assignments submitted later than 1, 2, 3, 4, and 5 days after the due date will be penalized by 10%, 20%, 30%, 40%, and 100%, respectively.
- Submit key parts of your code, results (e.g., plots, tables), and brief discussions as an MS Word or PDF document. You may generate this document using Jupyter Notebook/R Markdown or by taking snapshots.
- Ensure the entire code is submitted. If your code includes multiple files, please submit them as a single zip file along with the report.
- Homework assignments account for 65
- 4-6 "showcase" solutions to each assignment will be posted on Canvas.

Final Project

- Final Project contributes 15% towards the grade and will be due at 11:59 pm (Eastern Time) on December 16. NO late project will be accepted.
- List of final projects:
  - Case Study in Sentiment Analysis
  - Case Study in Fake News Detection
  - Case Study in Language Translation
  - Case Study in Text Generation
  - Case Study in Automatic Text Summarization
  - Case Study in Text Embedding Models
  - NLP for Healthcare: Medical Text Analysis
  - Case Study in Topic Modeling
  - Automatic Formatting and Styling of Text Documents
  - etc. (approx. 50 topics total to choose from)
- For the final project, students will submit:
  1. Working demo
  2. 7-15 pages of MS Word document
  3. 8-15 slides (PowerPoint or pdf)
  4. 10-15 minute video presentation (YouTube or similar)

Contents

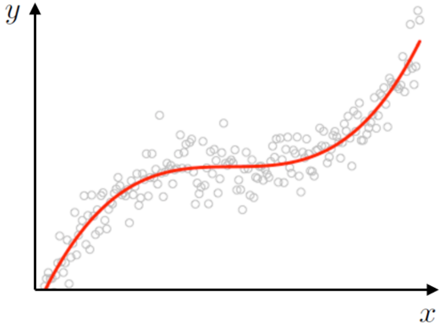
Dates of Interest

- The course starts, September 9
- Last day to change the credit status, September 10
- Course drop deadline for full-tuition refund, September 10
- Quiz 1 is due, September 16
- Assignment 1 is due, September 22
- Course drop deadline for half-tuition refund, September 17
- Withdrawal deadline, November 22
- **Final Project** is due, December 16, 11:59 pm (Eastern Time)

Contents

Example of ML: Linear Regression

Linear Regression on manually designed features u_1, u_2, and u_3:

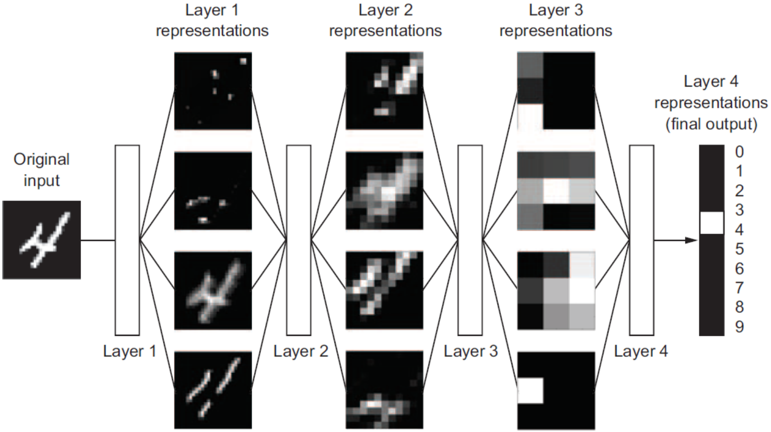


Model:

$$\hat{y} = w_0 + w_1 \cdot \underbrace{x}_{u_1} + w_2 \cdot \underbrace{x^2}_{u_2} + w_3 \cdot \underbrace{x^3}_{u_3}$$

Example of DL: Convolutional Neural Network (CNN)

Deep Learning as multistage learning of data representations:



Source: *Deep Learning* by F. Chollet

Example Feedforward Neural Network (FNN)

Let's consider a *Artificial Neural Network* which has two real-valued inputs (denoted by x_1 and x_2), one hidden layer that consists of two neurons (u_1 and u_2) with ReLU activation functions, and one output $\hat{y}$ with the ReLU activation function.

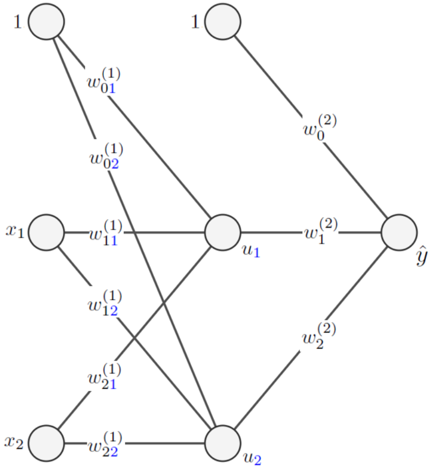
The explicit representation of this network is

input layer	hidden layer	output layer
x_1 x_2	$u_1 = f(w_{01}^{(1)} + w_{11}^{(1)} x_1 + w_{21}^{(1)} x_2)$ $u_2 = f(w_{02}^{(1)} + w_{12}^{(1)} x_1 + w_{22}^{(1)} x_2)$	$\hat{y} = f(w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2)$

Here, $f(x)$ denotes the rectified linear unit (ReLU) defined as follows:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$$

Example: Artificial NN with 2 inputs and 1 output (cont.)



Contents

Deep Feedforward Neural Network

Deep Neural Network (DNN) with

- n inputs
- M outputs
- $L-1$ hidden layers

is defined as:

$$\hat{\boldsymbol{y}} = f^{(L)}(f^{(L-1)}(\dots f^{(1)}(\boldsymbol{x}))),$$

where
$\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T \in \mathbb{R}^n$ are inputs,
$\hat{\boldsymbol{y}} = (\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M)^T \in \mathbb{R}^M$ are outputs,

and $f^{(1)}, \dots, f^{(L-1)}, f^{(L)}$ are vector-values functions.

Example: Regression with NNs

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}})$$

with

- $f^{(2)} : \mathbb{R}^H \mapsto \mathbb{R}$, i.e. $M = 1$,

  is used for *regression*.

Example: Regression with NNs

Keras:

```
import keras
from keras import models
from keras import layers

# number of inputs
n = 900

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(n,)))
model.add(layers.Dense(1, activation='relu'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                14416
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 17
=================================================================
Total params: 14,433
Trainable params: 14,433
Non-trainable params: 0
_________________________________________________________________
```

Example: Classification with NNs

NN

$$\hat{\boldsymbol{y}} = f^{(2)}(\underbrace{f^{(1)}(\boldsymbol{x})}_{\doteq \boldsymbol{u}})$$

with

- $f_m^{(2)}(\boldsymbol{u}) \geq 0$ for all $m \in \{1, 2, \ldots, M\}$ and $\boldsymbol{u} \in \mathbb{R}^H$ and $\sum_{m=1}^{M} f_m^{(2)}(\boldsymbol{u}) = 1$ for all $\boldsymbol{u} \in \mathbb{R}^H$

  is used for *classification*.

Example: Classification with NNs

Keras:

```
import keras
from keras import models
from keras import layers

# number of inputs
n = 900

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(n,)))
model.add(layers.Dense(2, activation='softmax'))

model.summary()
```

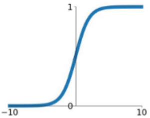
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 16)                14416
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 34
=================================================================
Total params: 14,450
Trainable params: 14,450
Non-trainable params: 0
_________________________________________________________________
```

Contents

Activation Functions

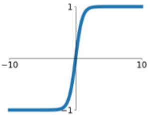
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



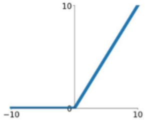
tanh

$\tanh(x)$



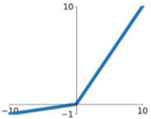
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

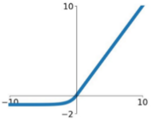


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions: Keras

```
model = models.Sequential()
model.add(layers.Dense(4, activation='sigmoid', input_shape=(4,)))
model.add(layers.Dense(4, activation='tanh'))
model.add(layers.Dense(4, activation=keras.layers.LeakyReLU(alpha=0.1)))
model.add(layers.MaxoutDense(4, nb_feature=2))
model.add(layers.Dense(4, activation=keras.layers.ELU(alpha=1.0)))
model.add(layers.Dense(1, activation='linear'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_8 (Dense)              (None, 4)                 20
_________________________________________________________________
maxout_dense_2 (MaxoutDense) (None, 4)                 40
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 5
=================================================================
Total params: 125
Trainable params: 125
Non-trainable params: 0
```

Contents

Loss Function

Assume we observe

$$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}),$$

where
$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ are n inputs and
$\boldsymbol{y} = (y_1, y_2, \ldots, y_M)^T$ are M outputs.

The prediction obtained with a supervised model (e.g., Neural Network) is denoted by

$$\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w}) = \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \boldsymbol{w}),$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^T$ are parameters of the model.

Loss Function

Assume we observe

$$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}),$$

where
$\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ are n inputs and
$\boldsymbol{y} = (y_1, y_2, \ldots, y_M)^T$ are M outputs.

The prediction obtained with a supervised model (e.g., Neural Network) is denoted by

$$\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w}) = \hat{\boldsymbol{y}}(\boldsymbol{x}^{(i)}; \boldsymbol{w}),$$

where $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^T$ are parameters of the model.

The loss incurred from incorrect prediction of $\boldsymbol{y}^{(i)}$ is

$$L^{(i)}(\boldsymbol{w}) = L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}}).$$

Loss Function: Prediction

Squared Error

What Loss can we choose in case of prediction? One of the most common loss functions is Squared Error.

If the output is scalar (i.e. $M = 1$), the Squared Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = (\hat{y}^{(i)} - y^{(i)})^2.$$

Loss Function: Prediction

Squared Error

What Loss can we choose in case of prediction? One of the most common loss functions is Squared Error.

If the output is scalar (i.e. $M = 1$), the Squared Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = (\hat{y}^{(i)} - y^{(i)})^2.$$

If the output is vector-valued (i.e. $M > 1$), the Squared Error Loss is then similarly defined as:

$$L^{(i)}(\boldsymbol{w}) = |\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}|^2 = \sum_{j=1}^{M} (\hat{y}_j^{(i)} - y_j^{(i)})^2.$$

Loss Function: Prediction

Absolute Error

An alternative to the Squared Error is Absolute Error.

If the output is scalar (i.e. $M = 1$), the Absolute Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = |\hat{y}^{(i)} - y^{(i)}|.$$

Loss Function: Prediction

Absolute Error

An alternative to the Squared Error is Absolute Error.

If the output is scalar (i.e. $M = 1$), the Absolute Error Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = |\hat{y}^{(i)} - y^{(i)}|.$$

If the output is vector-valued (i.e. $M > 1$), the Absolute Error Loss is then defined as:

$$L^{(i)}(\boldsymbol{w}) = |\hat{\boldsymbol{y}}^{(i)} - \boldsymbol{y}^{(i)}| = \left(\sum_{j=1}^{M} (\hat{y}_j^{(i)} - y_j^{(i)})^2 \right)^{\frac{1}{2}}.$$

Loss Function: Classification

Cross-Entropy

What Loss can we choose in case of classification? One of the most common loss functions is Cross-Entropy.

If there are two classes (i.e. $M = 2$), the Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\left(y_1^{(i)} \ln \hat{y}_1^{(i)} + y_2^{(i)} \ln \hat{y}_2^{(i)}\right).$$

Loss Function: Classification

Cross-Entropy

What Loss can we choose in case of classification? One of the most common loss functions is Cross-Entropy.

If there are two classes (i.e. $M = 2$), the Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\left(y_1^{(i)} \ln \hat{y}_1^{(i)} + y_2^{(i)} \ln \hat{y}_2^{(i)}\right).$$

If there are multiple classes (i.e. $M > 2$), the (Multi-Class) Cross-Entropy Loss is defined as follows:

$$L^{(i)}(\boldsymbol{w}) = -\sum_{j=1}^{M} y_j^{(i)} \ln \hat{y}_j^{(i)}.$$

Contents

Objective (Cost) Function

Suppose we want to train a supervised model (e.g., Neural Network) using a set of observations:

$$(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), (\boldsymbol{x}_3, \boldsymbol{y}_3), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)$$

then we define the objective (or cost) function as mean loss:

$$J(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} L^{(i)}(\boldsymbol{w}),$$

where

$$L^{(i)}(\boldsymbol{w}) = L(\underbrace{\hat{\boldsymbol{y}}^{(i)}(\boldsymbol{w})}_{\text{prediction}}, \underbrace{\boldsymbol{y}^{(i)}}_{\text{observed}})$$

is the loss associated with a single observation i as defined earlier.

Objective (Cost) Function

The list of the most common cost functions:

- Mean Squared Error:

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{M}(\hat{y}_j^{(i)} - y_j^{(i)})^2$$

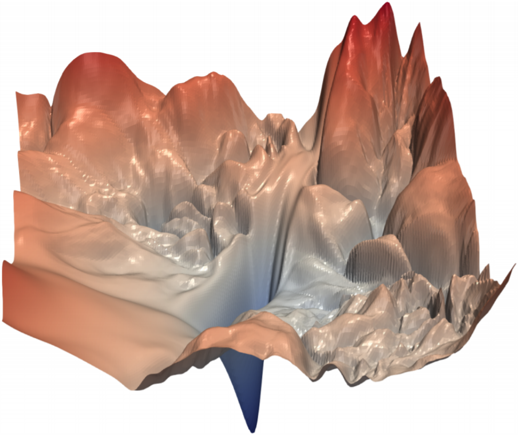
- Mean Absolute Error:

$$J(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{j=1}^{M}(\hat{y}_j^{(i)} - y_j^{(i)})^2\right)^{\frac{1}{2}}$$

- Cross-Entropy:

$$J(\boldsymbol{w}) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{M} y_j^{(i)} \ln \hat{y}_j^{(i)}$$

Cost Function Landscape



Source: *Visualizing the Loss Landscape of Neural Nets* by Hao Li et al., 2017

Keras: Classification Example

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(layers.Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_36 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_37 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

```
nepochs = 35
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')

history = model.fit(X_train, y_train,
          batch_size=128, epochs=nepochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

Keras: Loss Functions

More loss functions available in Keras:

mean_squared_error
mean_absolute_error
mean_absolute_percentage_error
mean_squared_logarithmic_error
squared_hinge
hinge
categorical_hinge
logcosh
categorical_crossentropy
sparse_categorical_crossentropy
binary_crossentropy
kullback_leibler_divergence
poisson
cosine_proximity

The complete list can be found at `https://keras.io/losses/`

Contents

Example of Forward Propagation/Backpropagation

Backpropagation:

Given weights $\boldsymbol{w}$, inputs x_1, x_2, and $z_1^{(1)}$, $z_2^{(1)}$, u_1, u_2, $\hat{y}$, compute

- Error associated with the output layer:

$$\varepsilon^{(2)} \doteq \frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left[(\hat{y} - y)^2 \right] = 2(\hat{y} - y)$$

- Errors associated with the hidden layer:

$$\varepsilon_h^{(1)} \doteq \frac{\partial L}{\partial u_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u_h} = \varepsilon^{(2)} f'(z^{(2)}) w_h^{(2)}, \quad h = 1, 2.$$

Example of Forward Propagation/Backpropagation

Computation of $\nabla L(\boldsymbol{w})$:

- Partial derivatives of the loss function with respect to weights in the output layer:

$$\frac{\partial L}{\partial w_h^{(2)}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_h^{(2)}} = \varepsilon^{(2)} \frac{\partial}{\partial w_h^{(2)}} \left[f(\underbrace{w_0^{(2)} + w_1^{(2)} u_1 + w_2^{(2)} u_2}_{z^{(2)}}) \right] = \varepsilon^{(2)} f'(z^{(2)}) u_h,$$

  where $h = 0, 1, 2$.

- Partial derivatives of the loss function with respect to weights in the hidden layer:

$$\frac{\partial L}{\partial w_{jh}^{(1)}} = \frac{\partial L}{\partial u_h} \frac{\partial u_h}{\partial w_{jh}^{(1)}} = \varepsilon_h^{(1)} \frac{\partial}{\partial w_{jh}^{(1)}} \left[f(\underbrace{w_{0h}^{(1)} + w_{1h}^{(1)} x_1 + w_{2h}^{(1)} x_2}_{z_h{}^{(1)}}) \right] = \varepsilon_h^{(1)} f'(z_h{}^{(1)}) x_j,$$

  for each $j = 0, 1, 2$ and $h = 1, 2$. Here, we define $x_0 \doteq 1$.

Example of Forward Propagation/Backpropagation

The Stochastic Gradient Descent (SGD) update of the weights using learning rate α:

$$\mathbf{w} := \mathbf{w} - \alpha \nabla L,$$

where $\nabla L \doteq \Big(\underbrace{\frac{\partial L}{\partial w_{01}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(1)}}, \frac{\partial L}{\partial w_{21}^{(1)}}, \frac{\partial L}{\partial w_{02}^{(1)}}, \frac{\partial L}{\partial w_{12}^{(1)}}, \frac{\partial L}{\partial w_{22}^{(1)}}}_{\text{hidden layer}}, \underbrace{\frac{\partial L}{\partial w_{0}^{(2)}}, \frac{\partial L}{\partial w_{1}^{(2)}}, \frac{\partial L}{\partial w_{2}^{(2)}}}_{\text{output layer}} \Big)^T.$

Therefore,

$$\begin{aligned}
\mathbf{w} &:= \mathbf{w} - \alpha \nabla L \\
&= \Big(\underbrace{w_{01}^{(1)}, w_{11}^{(1)}, w_{21}^{(1)}, w_{02}^{(1)}, w_{12}^{(1)}, w_{22}^{(1)}}_{\text{hidden layer}}, \underbrace{w_{0}^{(2)}, w_{1}^{(2)}, w_{2}^{(2)}}_{\text{output layer}} \Big)^T \\
&- \alpha \Big(\underbrace{\frac{\partial L}{\partial w_{01}^{(1)}}, \frac{\partial L}{\partial w_{11}^{(1)}}, \frac{\partial L}{\partial w_{21}^{(1)}}, \frac{\partial L}{\partial w_{02}^{(1)}}, \frac{\partial L}{\partial w_{12}^{(1)}}, \frac{\partial L}{\partial w_{22}^{(1)}}}_{\text{hidden layer}}, \underbrace{\frac{\partial L}{\partial w_{0}^{(2)}}, \frac{\partial L}{\partial w_{1}^{(2)}}, \frac{\partial L}{\partial w_{2}^{(2)}}}_{\text{output layer}} \Big)^T
\end{aligned}$$

Contents

SGD, mini-batch GD, and GD Optimization: 'sgd'

The SGD, mini-batch GD, and GD Optimization (with learning rate α) are all defined as follows:

$$\boldsymbol{w} := \boldsymbol{w} - \alpha \underbrace{\frac{1}{s} \sum_{i=1}^{s} \nabla L^{(i)}(\boldsymbol{w})}_{\approx \nabla J(\boldsymbol{w})},$$

where $L^{(i)}(\boldsymbol{w})$ is based on one observation i and

- $s = 1$ in case of Stochastic Gradient Descent (SGD)
- $1 < s < m$ in case of mini-batch Gradient Descent (mini-batch GD)
- $s = m$ in case of Gradient Descent (GD)

Here, m denotes the total number of observations in the data set.

SGD, mini-batch GD, and GD Optimization

Example: Classification via mini-batch GD with $s = 128$ and $\alpha = 0.01$.

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(layers.Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_14"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_36 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_37 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

```
nepochs = 35
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='sgd')

history = model.fit(X_train, y_train,
          batch_size=128, epochs=nepochs,
          verbose=1,
          validation_data=(X_test, y_test))
```

SGD, mini-batch GD, and GD Optimization

Example: Classification via mini-batch GD with $s = 128$ and $\alpha = 0.05$.

```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(layers.Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_35 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_36 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_37 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
```

```
nepochs = 35
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
              optimizer=keras.optimizers.SGD(lr=0.05))

history = model.fit(X_train, y_train,
          batch_size=128, epochs=nepochs,
          verbose=1,
          validation_data=(X_test, y_test))
```