Mathematical techniques in data science

Lecture 20: Back-propagation

April 10th, 2019

Schedule

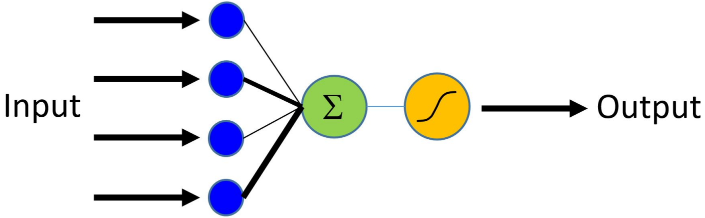
Week	Chapter
1	Chapter 2: Intro to statistical learning
3	Chapter 4: Classification
4	Chapter 9: Support vector machine and kernels
5, 6	Chapter 3: Linear regression
7	Chapter 8: Tree-based methods + Random forest
8	
9	**Neural networks**
12	PCA → Manifold learning
11	Clustering: K-means → Spectral Clustering
10	Bayesian methods + UQ
13	Reinforcement learning/Online learning/Active learning
14	Project presentation

Feed-forward neural networks

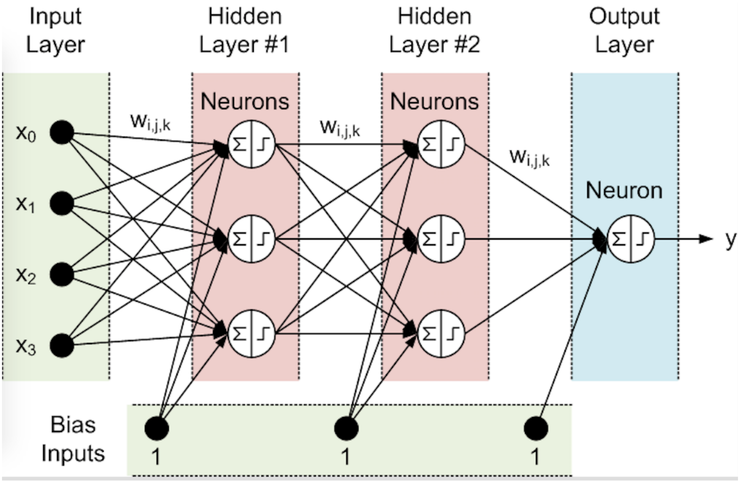
- Structure:
  - Graphical representation
  - Activation functions
  - Loss functions
- Training:
  - Stochastic gradient descent
  - Back-propagation

Feed-forward neural networks

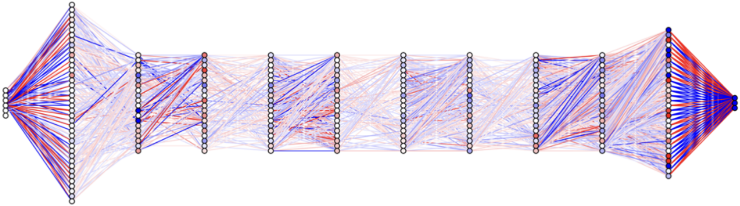
Logistic neuron



Feed-forward neural networks

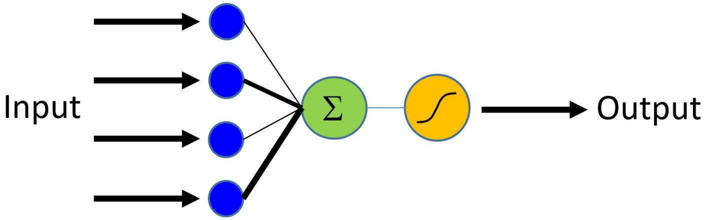


Feed-forward neural networks



Activation functions

Activation functions



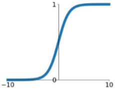
If we do not apply an activation function, then the output signal would simply be a simple linear function of the input signals

Activation functions

Activation Functions

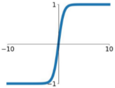
Sigmoid

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



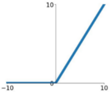
tanh

$\tanh(x)$



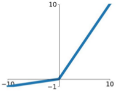
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

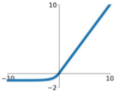


Maxout

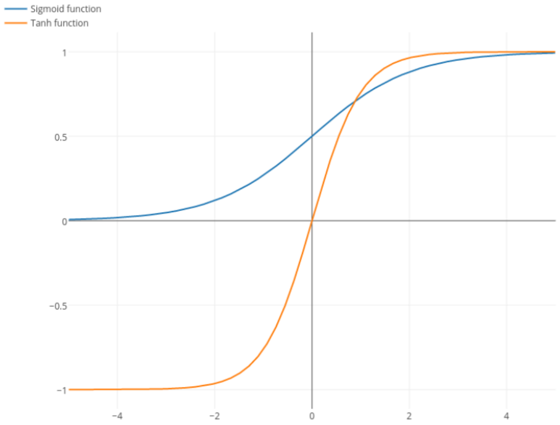
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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

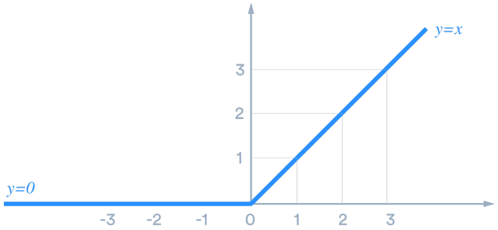


Hyperbolic tangent



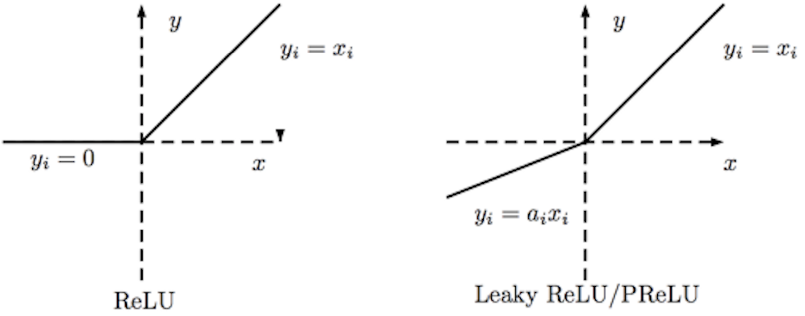
Issue: vanishing gradient problem

Rectified linear unit (ReLU)

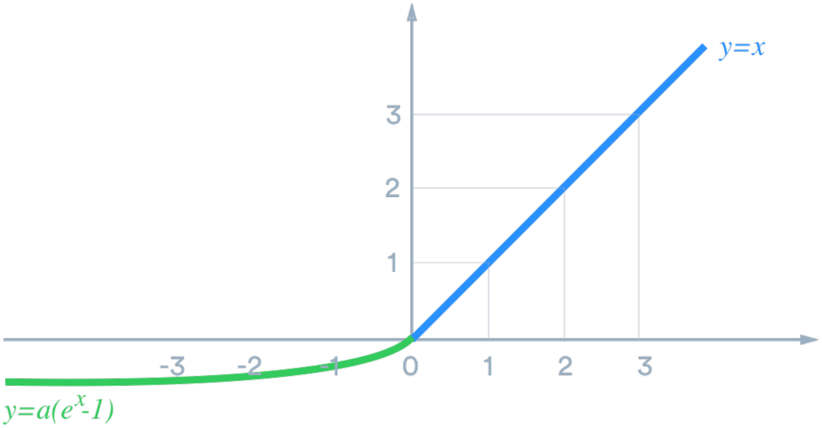


- Advantages: model sparsity, cheap to compute, partially address the vanishing gradient problem
- Issue: Dying ReLU

Leaky relu



Exponential Linear Unit (ELU)



Module: tf.keras.activations

Functions

`deserialize(...)`

`elu(...)`: Exponential linear unit.

`exponential(...)`

`get(...)`

`hard_sigmoid(...)`: Hard sigmoid activation function.

`linear(...)`

`relu(...)`: Rectified Linear Unit.

`selu(...)`: Scaled Exponential Linear Unit (SELU).

`serialize(...)`

`sigmoid(...)`

`softmax(...)`: Softmax activation function.

`softplus(...)`: Softplus activation function.

Loss functions

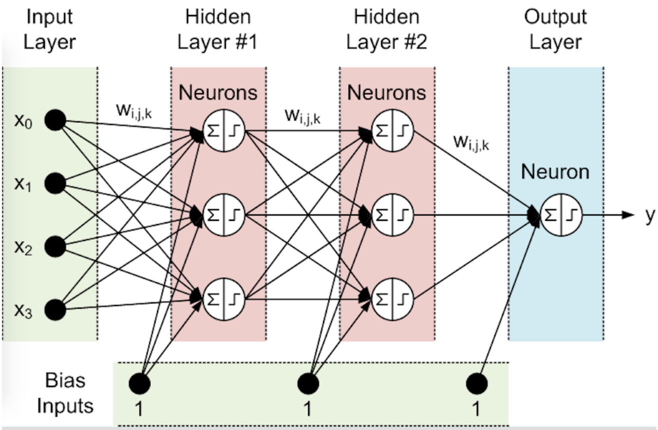
Supervised learning: standard setting

- Given: a sequence of label data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ sampled (independently and identically) from an unknown distribution $P_{X,Y}$
- The function h is an element of some space of possible functions $\mathcal{H}$, usually called the *hypothesis space*.
- In order to measure how well a function fits the training data, a *loss function*

  $$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^{\geq 0}$$

  is defined
- For training example (x_i, y_i) and a hypothesis h, the loss of predicting the value $h(x_i)$ is $L(y_i, h(x_i))$

Regression



For regression

$$L(w,x,y) = (y - h(w,x))^2, \quad \text{or,} \quad L(w,x,y) = |y - h(w,x)|$$

Classification: cross-entropy

Code

```
def CrossEntropy(yHat, y):
    if y == 1:
      return -log(yHat)
    else:
      return -log(1 - yHat)
```

Math

In binary classification, where the number of classes M equals 2, cross-entropy can be calculated as:

$$-(y\log(p) + (1-y)\log(1-p))$$

If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

Note: Here $y_{o,c}$ is the 0-1 label and $p_{o,c}$ is the predicted probability for the observation o is of class c, respectively

Module: tf.keras.losses

Classes

`class BinaryCrossentropy` : Computes the binary cross entropy loss between the labels and predictions.

`class CategoricalCrossentropy` : Computes categorical cross entropy loss between the `y_true` and `y_pred` .

`class MeanAbsoluteError` : Computes the mean of absolute difference between labels and predictions.

`class MeanAbsolutePercentageError` : Computes the mean absolute percentage error between `y_true` and `y_pred` .

`class MeanSquaredError` : Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError` : Computes the mean squared logarithmic error between `y_true` and `y_pred` .

Stochastic gradient descent

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

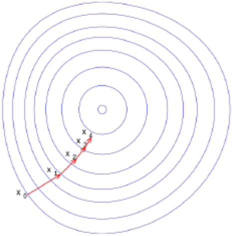


Figure: Gradient Descent. Source: http://en.wikipedia.org/wiki/Gradient_descent

Stochastic gradient descent

- Recall that the empirical risk function has the form

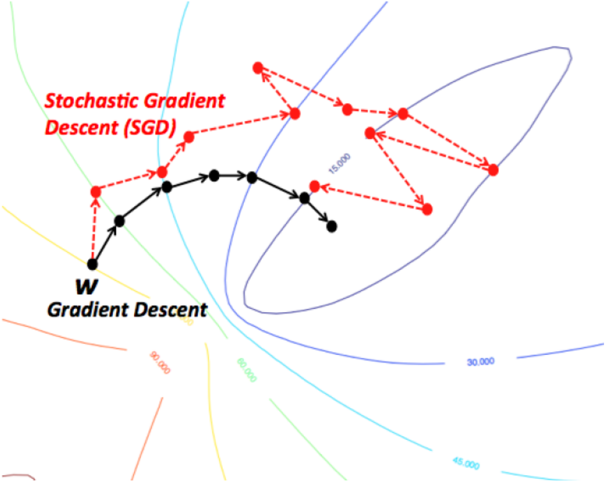
$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{N} L(w, x_i, y_i)$$

- Mini-batch stochastic gradient descent
  - randomly shuffle examples in the training set, divide them into k mini-batches of data of size m
  - for each batch I_i (i=1, ..., k), approximate the empirical risk by

    $$\hat{\mathcal{L}}(w) = \frac{1}{m} \sum_{j \in I_i} L(w, x_j, y_j)$$

    and update w by gradient descent
  - Repeat until an approximate minimum is obtained

Stochastic gradient descent (SGD)



Stochastic gradient descent: teminology

- Mini-batch stochastic gradient descent
  - randomly shuffle examples in the training set, divide them into k mini-batches of data of size m
  - for each batch I_i (i=1, ..., k), approximate the empirical risk by

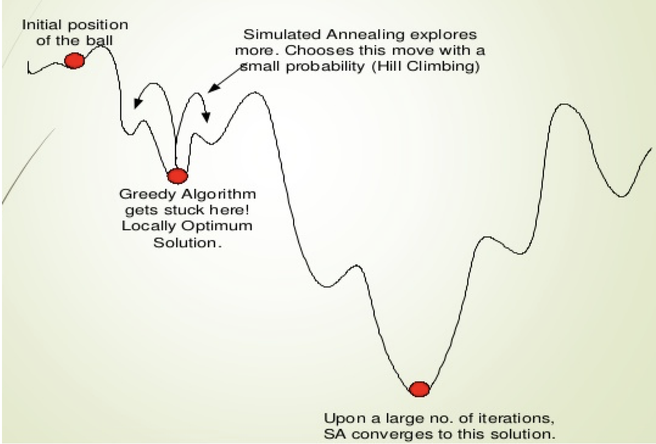
$$\hat{\mathcal{L}}(w) = \frac{1}{m} \sum_{j \in I_i} L(w, x_j, y_j)$$

    and update w by gradient descent
  - Repeat these steps M times
- Terminology:
  - m: batch-size
  - k: iteration
  - M: number of epochs

Stochastic gradient descent

- Gradient descent converges to the local minimum, and the fluctuation is small
- SGD's fluctuation is large, but enables jumping to new/better local minima

Related concept: Simulated annealing



Automatic diffierentiation

Stochastic gradient descent

- The most computationally heavy part in the training of a neural net is to compute

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}}$$

- Numerical differentiation is not realistic, and symbolic differentiation is impossible

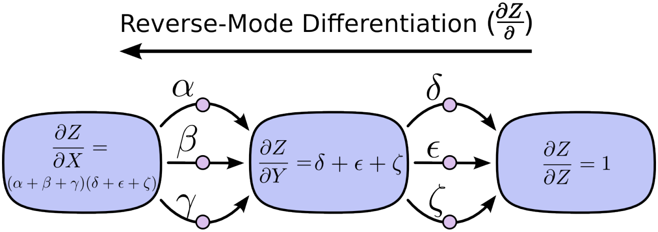
- Assume that

$$y = f(g(h(x)))$$

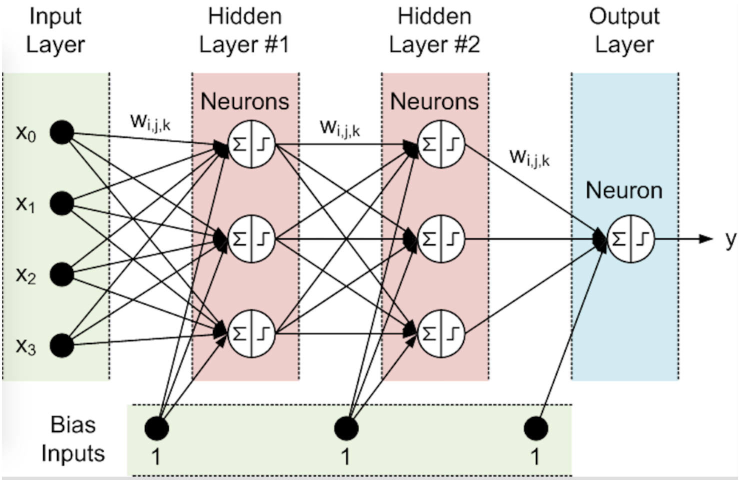
- Denote $x = u_0,\ h(u_0) = u_1,\ g(u_1) = u_2,\ f(u_2) = u_3 = y$, then

$$\frac{dy}{du_i} = \frac{dy}{du_{i+1}} \frac{du_{i+1}}{du_i}$$

Automatic differentiation



Feed-forward neural networks



Back-propagation

- Advantage: The cost to compute the partial derivatives with respect to all parameters are just twice the cost of a forward evaluations
- Drawback: The functions used to describe the network (activation functions and loss functions) needs to belong to the class of functions supported by the computational platform