

An Introduction to Deep Learning

Clint P. George

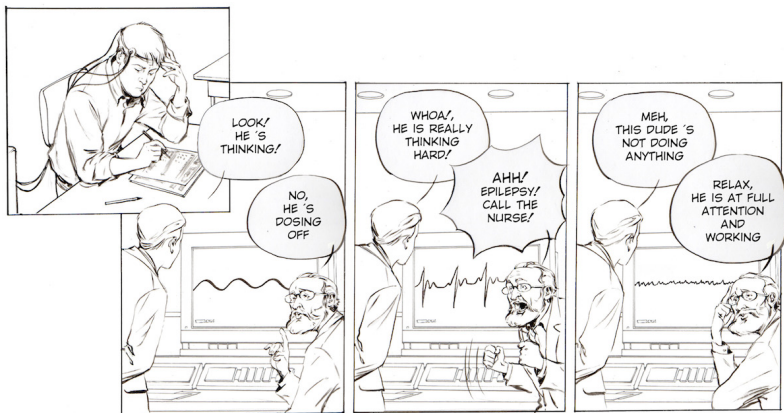
University of Florida Informatics Institute

March 14, 2017

Outline

- ① Introduction to feed-forward networks
- ② Relation to logistic regression
- ③ Notes on implementation
- ④ Illustration using synthetic and real data

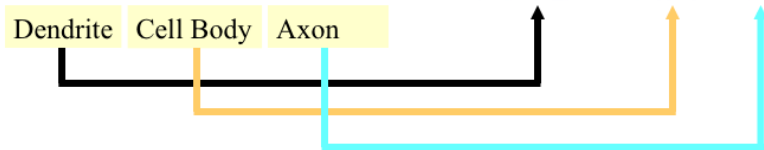
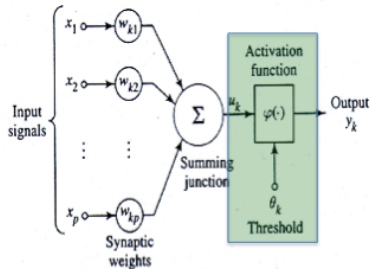
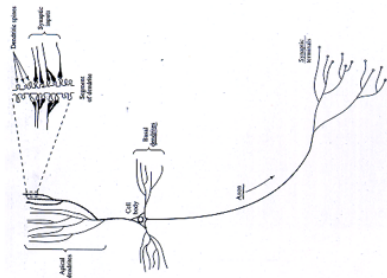
Motivation



The idea: Human intelligence may be due to a learning algorithm.
We aim to build algorithms that mimic the brain.¹

¹image: <https://backyardbrains.com/experiments/EEG>

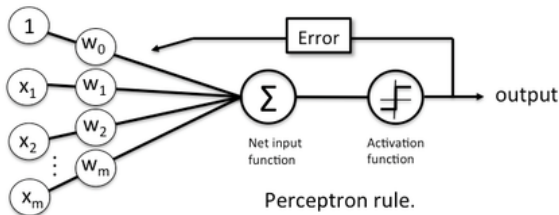
Imitate neurons in the brain: Artificial Neurons



Artificial Neuron (AN): input, weights, and output

Activation functions

The Perceptron (Rosenblatt et al. 1957 & 1962) computes a step function as an activation function.



$$\text{step}(z) = \begin{cases} 1 & z \geq t \\ 0 & z < t \end{cases}, \text{ where } t \text{ is a threshold}$$

As each input is applied to the perceptron its output is compared to the target. To keep the output closer to the target the **learning rule** adjusts the network parameters.

What can a single AN compute?

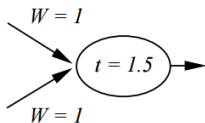
The Perceptron output is given by $y = \text{step}(b + \sum_{j=1}^p w_j x_j)$.

Perceptron can divide the input space into two regions.

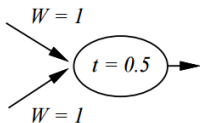
The decision boundary is given by: $b + \sum_{j=1}^p w_j x_j = 0$.

Perceptron can learn to classify any linearly separable set of inputs—convergence theorem (Rosenblatt 1962)

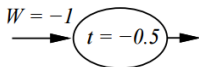
Boolean functions



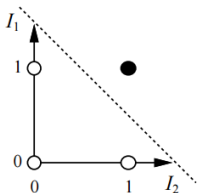
AND



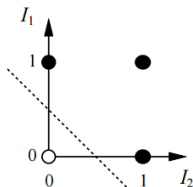
OR



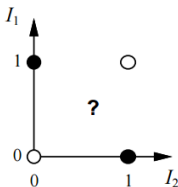
NOT



(a) I_1 and I_2



(b) I_1 or I_2



(c) I_1 xor I_2

Examples² of linearly separable and non separable problems

²Veloso, 2001

Learning the XOR function

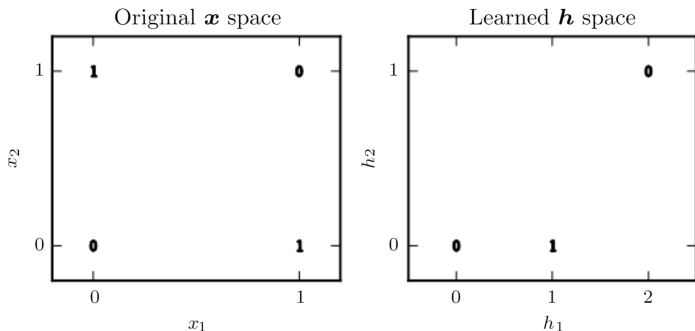
Consider this as a regression problem and use the MSE loss function:

$$\text{MSE}(\theta) = \frac{1}{4} \sum_{\mathbf{x} \in \mathcal{X}} (y - f_{\theta}(\mathbf{x}))^2$$

where $\theta = (\mathbf{w}, b)$ and $f_{\theta}(\mathbf{x}) = b + \sum_{j=1}^p w_j x_j$ —a linear model.

Using the normal equations we can minimize $\text{MSE}(\theta)$ w.r.t. \mathbf{w} and b in closed form. It gives $\mathbf{w} = 0$ and $b = .5$ —This gives the model output .5 everywhere.

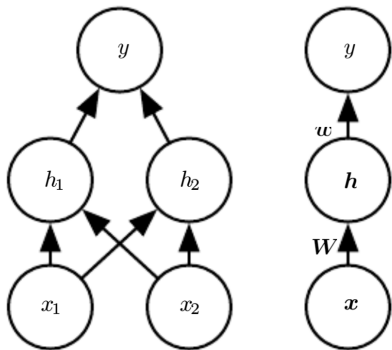
Learning the XOR function



- When $x_1 = 0$, the model's output must increase as x_2 increases
- When $x_1 = 1$, the model's output must decrease as x_2 increases
- A linear model applies a fixed coefficient w_2 to x_2 . It cannot use the value of x_1 to change the coefficient w_2 on x_2 and cannot solve this problem.

Intuition behind multilayer neural network

One way to solve the XOR problem is to transform the input by introducing a feed-forward network:



The complete model will then be, in a function form:

$$f(\mathbf{x}; \mathbf{W}, \mathbf{w}, b, c) = f^{(2)} \left(f^{(1)}(\mathbf{x}; \mathbf{W}, c); \mathbf{w}, b \right)$$

What function should $f^{(1)}$ be?

We consider $f^{(2)}$ as a linear function.

We can write the hidden layer output as

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) = g(\mathbf{W}^\top \mathbf{x} + \mathbf{c})$$

What function should $f^{(1)}$ be?

We consider $f^{(2)}$ as a linear function.

We can write the hidden layer output as

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) = g(\mathbf{W}^T \mathbf{x} + \mathbf{c})$$

If $f^{(1)}$ is also linear, then the network as a whole would remain a linear function of its input.

What function should $f^{(1)}$ be?

We consider $f^{(2)}$ as a linear function.

We can write the hidden layer output as

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) = g(\mathbf{W}^T \mathbf{x} + \mathbf{c})$$

So we must use a nonlinear activation function for g . A popular nonlinear activation function g is rectified linear unit (ReLU)

$$g(z) = \max\{0, z\}$$

A feed-forward network solution to XOR

We wish the network to perform well on the four cases

$$\mathbf{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

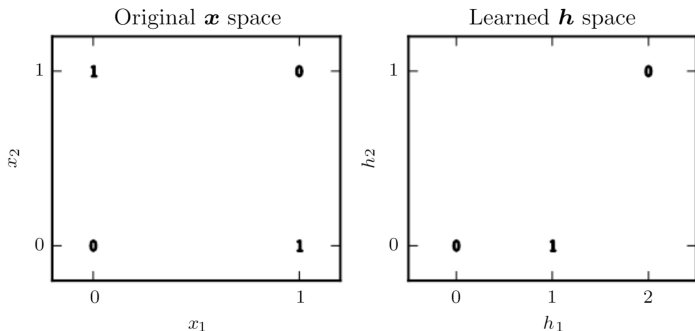
We can specify a two-layer network solution to XOR as

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

The complete network is given by

$$f(\mathbf{x}; \mathbf{W}, \mathbf{w}, b, \mathbf{c}) = \mathbf{w}^T \max\{0, \mathbf{W}^T \mathbf{x} + \mathbf{c}\} + b$$

A feed-forward network solution to XOR



In the proposed network, the nonlinear hidden layer has mapped both $\mathbf{x} = [1, 0]$ and $\mathbf{x} = [0, 1]$ to a single point in feature space, $\mathbf{h} = [1, 0]$.

A linear model can now describe the function as increasing in h_1 and decreasing in h_2 .

Activation functions

Modern ANs use a variety of activation functions that are smoother than the step function.

Linear - no input squashing

$$y = x$$

Logistic sigmoid - squash input into $[0, 1]$

$$y = \text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$

Hyperbolic tangent - squash input into $[-1, 1]$

$$y = \tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

Rectified linear unit

$$y = \max\{0, x\}$$

Relation to logistic regression

Let $p(y = 1 \mid X = \mathbf{x}) = p(\mathbf{x}; \mathbf{w})$ be the conditional probability that a particular sample belongs to class 1 given its predictors \mathbf{x} . We write the logistic regression model as

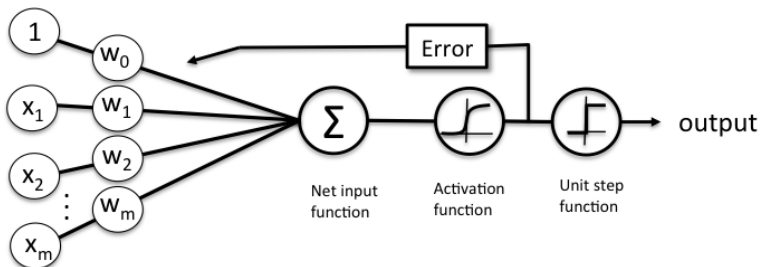
$$\text{logit}(p(\mathbf{x}; \mathbf{w})) = w_0 + \mathbf{w}^\top \mathbf{x}$$

Solving for $p(\mathbf{x}; \mathbf{w})$ gives

$$p(\mathbf{x}; \mathbf{w}) = \text{sigmoid}(w_0 + \mathbf{w}^\top \mathbf{x})$$

To minimize misclassification rate, one should predict $y = 1$ when $p \geq .5$, and vice versa.—i.e. guess 1 whenever $w_0 + \mathbf{w}^\top \mathbf{x} \geq 0$ and 0 otherwise. So logistic regression gives a linear classifier.

Relation to logistic regression



Schematic of a logistic regression classifier.

3

Learning parameters w and b : maximum likelihood estimation

- Perceptron algorithm: online and error-driven.
- Logistic regression: batch algorithms—e.g. gradient descent, limited-memory BFGS, or online algorithms—stochastic gradient descent.

³http://rasbt.github.io/mlxtend/user_guide/classifier/LogisticRegression/

Gradient-based learning

The nonlinearity of a neural network causes most interesting loss functions to become non-convex

- Optimization procedure – using iterative, gradient-based optimizers that drive the cost function to a very low value
- Cost function, $C(\theta)$ – they are more or less the same as those for other parametric models, such as linear models

Cost function

We define the loss functional $\mathcal{L}(f_\theta, z)$ based on the network, e.g., squared error, the negative conditional log-likelihood. Here, $z = (x, y)$ and $f_\theta(x)$ is the predictive function for y given θ .

We define the cost function as

$$C(\theta) = \int \mathcal{L}(f_\theta, z) P(z) dz$$

We typically write this as an average—*training loss*

$$C(\theta) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f_\theta, z)$$

Gradient descent (GD) algorithm⁴

We find a θ that minimizes the cost

- By solving $\frac{\partial C(\theta)}{\partial \theta} = 0$ we can find the minima, maxima, and saddle points.
- In general, we cannot find the solutions of this equation. So we seek numerical optimization methods
- *local descent*: iteratively modify θ so as to decrease $C(\theta)$, until we reach a local minima

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \frac{\partial C(\theta^{(t)})}{\partial \theta^{(t)}},$$

ϵ is the learning rate

⁴<http://www.iro.umontreal.ca/~pift6266/H10/notes/gradient.html>

Stochastic gradient descent (SGD) algorithm

- We use the fact that $C(\theta)$ is an average over i.i.d. samples
- Make updates much often

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \frac{\partial \mathcal{L}(\theta^{(t)}, z)}{\partial \theta^{(t)}},$$

z is an the next sample from the training set.—It can be implemented online

- The update direction is a random variable whose expectation is the true gradient of interest.

Output units

The output layer provides additional transformation from the hidden features \mathbf{h} to complete the network's indented task.

Linear units for Gaussian output distributions: the output units based on an affine transformation with no nonlinearity

- Given hidden features \mathbf{h} , we define outputs $f_{\theta}(\mathbf{x}) = \mathbf{w}^{\top} \mathbf{h} + b$
- It's typically used to produce the mean of a conditional Gaussian distribution $p(y|\mathbf{x}, \theta) = \mathcal{N}(f_{\theta}(\mathbf{x}), \mathbf{I})$

Cost function: the negative log-likelihood

The neural network defines a conditional distribution $p(y | \mathbf{x}, \theta)$.

Suppose we use MLE for parameter estimation. Then, it's natural to use the cost function as the negative log-likelihood:

$$C(\theta) = -\mathbb{E}_{\mathbf{x}, y} \log p_{\text{model}}(y | \mathbf{x}, \theta)$$

Example

if $p_{\text{model}}(y | \mathbf{x}, \theta) = \mathcal{N}(f_{\theta}(\mathbf{x}), I)$, then we have

$$C(\theta) = \frac{1}{2} \mathbb{E}_{\mathbf{x}, y} \|y - f_{\theta}(\mathbf{x})\|^2 + \text{const.}$$

Output units

Sigmoid units for Bernoulli output distributions: e.g. a binary classification problem

Given hidden features \mathbf{h} , we define the output as

$$\mu = \text{sigmoid}(\eta) = \text{sigmoid}(\mathbf{w}^T \mathbf{h} + b)$$

Bernoulli distribution and logistic sigmoid

For a Bernoulli distribution with $y \in \{0, 1\}$ that represents either success or failure, and $0 \leq \mu \leq 1$ representing the probability of success, we have

$$p(y | \mu) = \mu^y (1 - \mu)^{(1-y)} \quad (1)$$

$$= \exp\{y \log \mu + (1 - y) \log(1 - \mu)\} \quad (2)$$

$$= \exp\left\{y \log \frac{\mu}{1 - \mu} + \log(1 - \mu)\right\} \quad (3)$$

Comparing this expression with the density of an exponential family distribution $p(y | \eta) = h(x) \exp\{\eta \mathbf{T}(y) - \mathbf{A}(\eta)\}$,

Bernoulli distribution and logistic sigmoid

For a Bernoulli distribution with $y \in \{0, 1\}$ that represents either success or failure, and $0 \leq \mu \leq 1$ representing the probability of success, we have

$$p(y | \mu) = \mu^y (1 - \mu)^{(1-y)} \quad (1)$$

$$= \exp\{y \log \mu + (1 - y) \log(1 - \mu)\} \quad (2)$$

$$= \exp\left\{y \log \frac{\mu}{1 - \mu} + \log(1 - \mu)\right\} \quad (3)$$

Comparing this expression with the density of an exponential family distribution $p(y | \eta) = h(x) \exp\{\eta T(y) - A(\eta)\}$, where η is the natural parameter, $T(y)$ is the sufficient statistic, $A(\eta)$ is the log partition function, and $h(x)$ is a normalizing constant,

Bernoulli distribution and logistic sigmoid

For a Bernoulli distribution with $y \in \{0, 1\}$ that represents either success or failure, and $0 \leq \mu \leq 1$ representing the probability of success, we have

$$p(y | \mu) = \mu^y (1 - \mu)^{(1-y)} \quad (1)$$

$$= \exp\{y \log \mu + (1 - y) \log(1 - \mu)\} \quad (2)$$

$$= \exp\left\{y \log \frac{\mu}{1 - \mu} + \log(1 - \mu)\right\} \quad (3)$$

Comparing this expression with the density of an exponential family distribution $p(y | \eta) = h(x) \exp\{\eta T(y) - A(\eta)\}$, we have

$$\eta = \log \frac{\mu}{1 - \mu} - \log \text{odds} \quad (4)$$

$$\rightarrow \mu = \text{sigmoid}(\eta) \quad (5)$$

Parameter estimation via MLE

We assume we have a training set $\mathcal{X} = (\mathbf{x}_i, y_i), i = 1, 2, \dots, n$. Our goal is to maximize the log likelihood:

$$\log p(\mathcal{X} | \theta) = \sum_{i=1}^n \log p(\mathbf{x}_i, y_i | \theta).$$

We factor the log likelihood into an unconditional term $p(\mathbf{x})$ that we ignore and a conditional term $p(y | \mathbf{x})$ that we focus.

Let $z_i = \mathbb{I}(y_i == 1)$ and $p_i = p(y_i = 1 | \mathbf{x}_i)$ —a logistic linear function of \mathbf{x}_i for a single layer network. We can write the conditional density of the dataset as

$$\mathcal{L} = \prod_{i=1}^n p_i^{z_i} (1 - p_i)^{1-z_i}$$

Parameter estimation via MLE

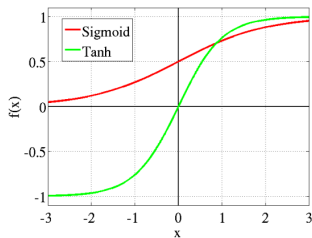
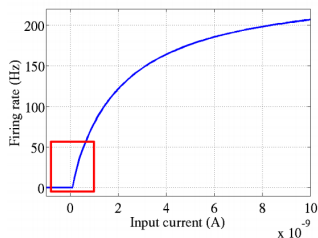
We wish to maximize the log likelihood:

$$\log \mathcal{L} = \sum_{i=1}^n z_i \log p_i + (1 - z_i) \log (1 - p_i)$$

The negative of this log likelihood is a *cross entropy* between the indicator variables z and the posterior probabilities p .

This also shows that the cross entropy is a natural cost function for binary classification problem (Jordan 1995)

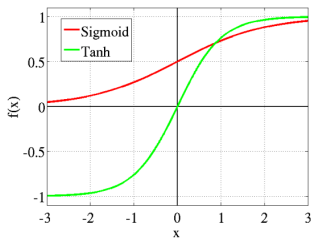
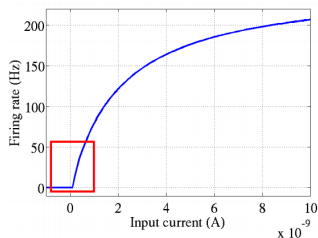
Hidden units: sigmoid and hyperbolic tangent⁵



Common neural activation function motivated by biological data (left) and sigmoid and hyperbolic tangent (right).

⁵Glorot (2011)

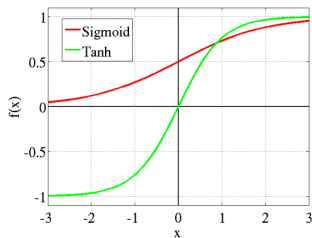
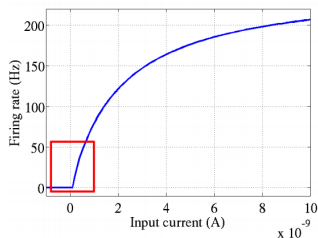
Hidden units: sigmoid and hyperbolic tangent⁵



During training sigmoidal units saturate across most of their domain—they saturate to a high value when z is very positive and vice versa. It can make gradient-based learning very difficult.

⁵Glorot (2011)

Hidden units: sigmoid and hyperbolic tangent⁵



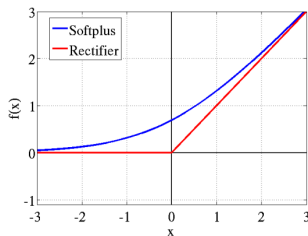
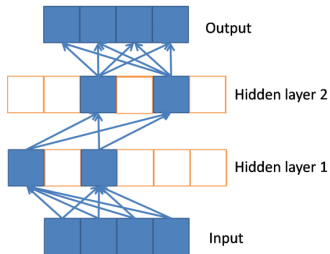
The hyperbolic tangent has a steady state at 0, hence is preferred from the optimization standpoint. It forces an antisymmetry around 0 which is absent in biological neurons.

⁵Glorot (2011)

Hidden units: ReLU

A popular choice of activation function for hidden units⁶

$$g(z) = \max\{0, z\}$$

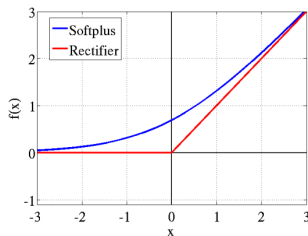
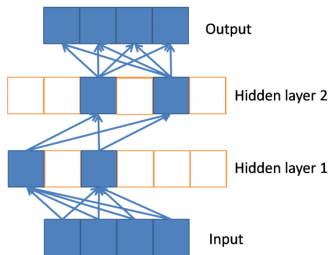


⁶Glorot (2011)

Hidden units: ReLU

A popular choice of activation function for hidden units⁶

$$g(z) = \max\{0, z\}$$



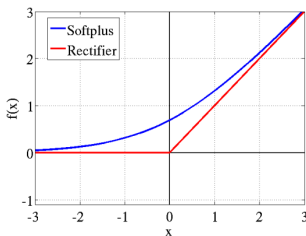
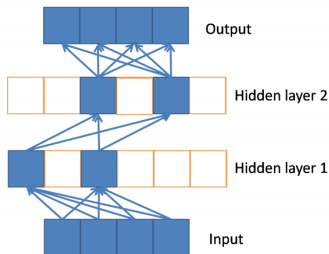
One can safely disregard nonlinearity (Goodfellow et al. 2016): “... neural network training algorithms do not usually arrive at a local minimum of the cost function, but instead merely reduce its value significantly”

⁶Glorot (2011)

Hidden units: ReLU

A popular choice of activation function for hidden units⁶

$$g(z) = \max\{0, z\}$$



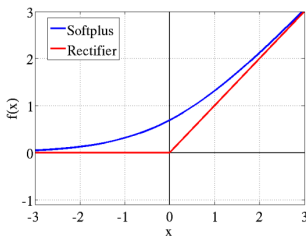
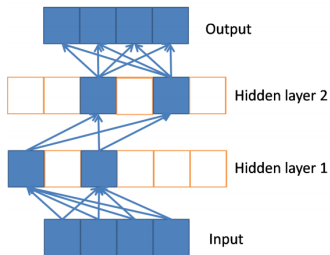
Non-linearity in the network comes from the path selection associated with individual neurons being active or not. Once this subset of neurons is selected, the output is a linear function of the input.

⁶Glorot (2011)

Hidden units: ReLU

A popular choice of activation function for hidden units⁶

$$g(z) = \max\{0, z\}$$



ReLU allows a network to easily obtain sparse representations.

⁶Glorot (2011)

An overview of back-propagation algorithm

Computing an analytical expression for the gradient is straightforward, but numerical evaluation of such an expression can be expensive over the network.—**backprop** gives an inexpensive procedure to evaluate this gradient.

backprop uses the chain rule of calculus: Suppose $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \rightarrow \mathbb{R}^n$, $f : \mathbb{R}^n \rightarrow \mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$, $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_i}$$

We can generalize this procedure to vectors and tensors in a multilayer neural network.

Notes on architectural considerations

In practice, the overall structure of the network is important: how many units it should have and how these units should be connected to each other.

Alternatives to feed-forward networks

- Convolutional Neural Networks — imitates human memory
- Auto Encoders — unsupervised learning and dimensionality reduction

Hands on experiments

Datasets

- A synthetic spiral dataset with multiple classes
- The MNIST data set with 0-9 handwritten characters

Algorithm: A basic back-propagation algorithm implementation with one hidden layer

Link to R scripts: <http://bit.ly/deep-learning-stats>

Links to serious implementations

Deep learning frameworks, which uses CPU and GPU

- TensorFlow with R
<https://rstudio.github.io/tensorflow/index.html>
- Theano with python
<http://deeplearning.net/software/theano>

Available R packages

- neuralnet
- deepnet
- h2o

Questions?