ML Reading Group

Neural Networks - Introduction

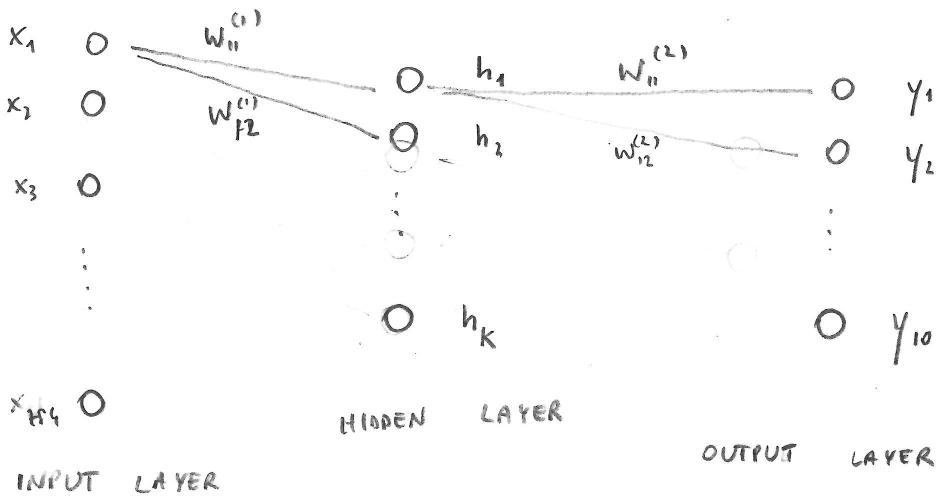
Reference Michael Nielsen

→ Basic concepts

→ Pointers to papers

neuralnetworksanddeeplearning.com

Architecture (simplest possible)



LEAVE SOME SPACE FOR FUTURE STUFF!

Task

→ Regression. Pick $f: \mathbb{R}^d \to \mathbb{R}^t$ and suppose we want to learn this function. This means we have access to n samples $(x(i), y(i))$ for $i \in \{1, \dots, n\}$, with $x(i) \in \mathbb{R}^d$, $y(i) \in \mathbb{R}^t$, s.t. $y(i) = f(x(i))$. Again, we have $\{(x(i), y(i))\}_{i=1}^{n}$ but we do not know f. Given $x \in \mathbb{R}^d$, we want to find a good approximation for $f(x)$.

Supervised learning task because we input and output.

→ Classification. Same as before but y represents the label of a discrete class. Classical example is MNIST (http://yann.lecun.com/exdb/mnist) in which we are given 60k labeled images. Each image is a 28 × 28 b/w picture of a handwritten digit and the label represents what digit it is :

$x \in \mathbb{R}^{784}$ and $y \in \{0, 1, 2, \dots, 9\}$.

Again supervised learning. Test set 10K images of which we want to find out the label.

→ Clustering. Group data into some number of groups (clusters) without labels. No labels ⇒ unsupervised learning.

→ Density estimation. Again unsupervised

Also semi-supervised learning in which part of data is labeled and part is not labeled. (labeled data is costly to generate, while unlabeled data is not).

Will focus on classification :

INPUT $x \in \mathbb{R}^{784}$ 784 nodes as input (1 per pixel)

OUTPUT $y \in \{0, 1, \dots, 9\}$ or more frequently

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \dots \quad \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

"0" "1" "2" "9"

$y \in \{0, 1\}^{10}$

10 nodes as output

Again on architecture

→ Feed forward. Allows signals to travel one way only : from input to output. No feedback or loops, the output of any layer does not affect the same layer.

→ Feedback / recurrent / interactive : signals travelling in both direction

by introducing loops in the network.

Will focus on FEEDFORWARD

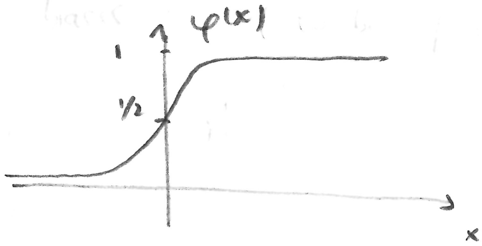
More on architecture

$$h_i^{(1)} = \varphi\Big(\sum_J W_{Ji}^{(1)} x_J + b_i^{(1)}\Big)$$

- $W_{iJ}^{(1)}$, $b_i^{(1)}$ weights and biases (to be found)
- φ ACTIVATION FUNCTION (given)

→ Non-linearity

→ $\varphi(x) = \dfrac{1}{1+e^{-x}}$ sigmoid



range $[0,1]$, historically popular

$$y_i = \varphi\Big(\sum_J W_{iJ}^{(2)} h_J + b_i^{(2)}\Big)$$

Find $W_{iJ}^{(1)}$, $W_{iJ}^{(2)}$, $b_i^{(1)}$, $b_i^{(2)}$ st.

COST FUNCTION

$$C(W,b) = \frac{1}{2n}\sum_{i,K}\Big(y_i^{(K)} - \overbrace{\varphi\Big(\sum_J W_{Ji}^{(2)}\varphi\Big(\sum_{J'} W_{J'J}^{(1)} x_{J'}^{(K)} + b_J^{(1)}\Big) + b_i^{(2)}\Big)}^{\hat{y}_i^{(K)}}\Big)^2$$

coordinates, samples

is minimized

→ MSE cost function

→ cross-entropy $C = -\dfrac{1}{n}\sum_{i,K}\Big[y_i^{(K)}\ln \hat{y}_i^{(K)} + (1-y_i^{(K)})\ln(1-\hat{y}_i^{(K)})\Big]$

= KL distance from $y^{(K)}$ to $\hat{y}^{(K)}$.

Minimize cost function by gradient descent and find weights and biases.

Types of gradient descent

→ Batch : the standard one. Compute ∇C and

$$\begin{bmatrix} W \\ b \end{bmatrix} \leftarrow \begin{bmatrix} W \\ b \end{bmatrix} - \gamma \nabla C$$

PROBLEM : n is typically large (60k for MNIST, 1.2 M for ImageNet) and we have to perform $\sum$ over n samples....

$$\nabla C \equiv \sum_{i=1}^{n} \nabla C^{(i)}$$

→ Stochastic : pick one sample at random, compute gradient, update and repeat. This means not to $\sum$ over samples.

$$\nabla C \equiv \nabla C^{(i)}$$

→ Mini-batch : divide the samples into mini-batches of size M = 100. For each mini-batch do batch gradient descent, update, go to the next mini-batch

$$\nabla C = \sum_{i=1}^{M} \nabla C^{(i)}$$

How well can we do?

Consider regression pb., i.e., learning a function f.

→ Representation : Do weights and biases that approximate well enough f even exist? Is the function f representable by the neural net?

YES, if the number of hidden nodes is large enough.

"Approximation by superposition of a sigmoidal function", Cybenko, 1989

"Universal approximation bounds for superpositions of a sigmoidal function", Barron, 1993.

suppose we can represent ...

→ learning error. After training, cost function does not go to 0.

Cost function is non-convex so no guarantees in principle!

"Provable bounds for learning some Deep Representations",

Arora, 2013.

Give an algorithm that provably learns a class of networks in poly running time. Key: sparsity. You start with a learnable function and you show that you can actually learn it.

"The loss surfaces of multilayer networks", Choromanska, 2014.

connection with spin-glass, structural barrier to learning.

→ Generalization error. Cost function for training set is small, but errors for test set are still large. Training set does not represent well test set.

"Stability and generalization", Bousquet, 2002

Stability ⟹ Generalization

Stability = if 2 inputs are close, then corresponding output are also close.

"Train faster, generalize better: stability of stochastic gradient descent", Hardt, 2016.

If training does not take too long ⟹ small generalization error.

Backpropagation Algorithm

→ introduced in 70's and popularized by

" Learning representations by back-propagating errors " Rumelhart , 1986

L layers , activation function $\varphi(x)$ for all layers (for simplicity)

$W_{JK}^{(\ell)}$ - weight of connection from K^{th} node in the $(\ell-1)^{th}$ layer to J^{th} node in the ℓ^{th} layer

$b_J^{(\ell)}$ - bias of the J^{th} neuron in the ℓ^{th} layer.

$$h_J^{(\ell)} = \varphi\Big(\overbrace{\sum W_{JK}^{(\ell)} a_K^{(\ell-1)} + b_J^{(\ell)}}^{z_J^{(\ell)}} \Big)$$

VECTORIZED FORM $$\underline{h}^{(\ell)} = \varphi\Big(\overbrace{\underline{\underline{W}}^{\ell}\, \underline{h}^{(\ell-1)} + \underline{b}^{(\ell)}}^{z^{(\ell)}} \Big)$$

applied component-wise

$$C = \frac{1}{2n} \sum_K \| \underline{y}(K) - \underline{h}^{(L)}(x(K)) \|^2$$

$$= \frac{1}{n} \sum_K \underbrace{C_K}$$

cost function with respect to the K-th training example

Suppose we have 1 training example.

Aim : compute $\dfrac{\partial C}{\partial W_{JK}^{(\ell)}}$, $\dfrac{\partial C}{\partial b_J^{(\ell)}}$

Define $\delta_J^{(\ell)} = \dfrac{\partial C}{\partial \underbrace{z_J^{(\ell)}}}$ with $h_J^{(\ell)} = \varphi(z_J^{(\ell)})$

input of J^{th} node in ℓ^{th} layer BEFORE activation function

LAST LAYER

$$\frac{\partial h_k^{(L)}}{\partial z_J^{(L)}} = 0 \quad \text{for } k \neq J$$

$$\delta_J^{(L)} \overset{\text{definition}}{=} \frac{\partial C}{\partial z_J^{(L)}} \overset{\text{chain rule}}{=} \sum_k \frac{\partial C}{\partial h_k^{(L)}} \frac{\partial h_k^{(L)}}{\partial z_J^{(L)}} = \frac{\partial C}{\partial h_J^{(L)}} \frac{\partial h_J^{(L)}}{\partial z_J^{(L)}} = \frac{\partial C}{\partial h_J^{(L)}} \varphi'(z_J^{(L)})$$

$z_J^{(L)}$ is easy to compute (forward pass of the network)

$\Downarrow$

$\varphi'(z_J^{(L)})$ also easy

$$\frac{\partial C}{\partial h_J^{(L)}} = h_J^{(L)} - y_J \quad \dots \text{ easy}$$

Hadamard product

$$\boxed{\delta^{(L)} = \nabla_{h^{(L)}} C \odot \varphi'(\underline{z}^{(L)})}$$

$$(a \odot b)_i = a_i \cdot b_i$$

$$\delta_J^{(\ell)} = \frac{\partial C}{\partial z_J^{(\ell)}} = \sum_k \frac{\partial C}{\partial z_k^{(\ell+1)}} \frac{\partial z_k^{(\ell+1)}}{\partial z_J^{(\ell)}} = \sum_k \frac{\partial z_k^{(\ell+1)}}{\partial z_J^{(\ell)}} \delta_k^{(\ell+1)} = *$$

$$z_k^{(\ell+1)} = \sum_J W_{kJ}^{(\ell+1)} \varphi(z_J^{(\ell)}) + b_k^{(\ell+1)}$$

$$* = \sum_k W_{kJ}^{(\ell+1)} \delta_k^{(\ell+1)} \varphi'(z_J^{(\ell)})$$

$$\boxed{\delta^{(\ell)} = (W^{(\ell+1)})^T \delta^{(\ell+1)} \odot \varphi'(z^{(\ell)})}$$

(backward pass of the network)

$$\frac{\partial z_k^{(l)}}{\partial b_J^{(l)}} = 0 \quad \text{for } k \neq J$$

$$\frac{\partial C}{\partial b_J^{(l)}} = \sum_k \frac{\partial C}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_J^{(l)}} = \frac{\partial C}{\partial z_J^{(l)}} \underbrace{\frac{\partial z_J^{(l)}}{\partial b_J^{(l)}}}_{=1} = \delta_J^{(l)} \cdot 1$$

$$z_J^{(l)} = \sum_k W_{Jk}\, \varphi(z_k^{(l-1)}) + b_J^{(l)}$$

$$\boxed{\frac{\partial C}{\partial b_J^{(l)}} = \delta_J^{(l)}}$$

$$\frac{\partial z_i^{(l)}}{\partial W_{JK}^{(l)}} = 0 \quad \text{for } i \neq J$$

weight from node k of layer $l-1$ to node J of layer l

$$\frac{\partial C}{\partial W_{JK}^{(l)}} = \sum_i \frac{\partial C}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial W_{JK}^{(l)}} = \frac{\partial C}{\partial z_J^{(l)}} \frac{\partial z_J^{(l)}}{\partial W_{JK}^{(l)}}$$

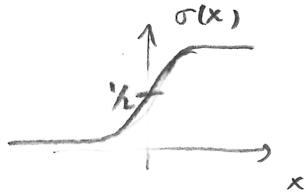
$$= \delta_J^{(l)}\, \varphi(z_K^{(l-1)}) = \delta_J^{(l)}\, h_K^{(l-1)}$$

$$\boxed{\frac{\partial C}{\partial W_{JK}^{(l)}} = \delta_J^{(l)}\, h_K^{(l-1)}}$$

write gradient as function of δ's

More on activation functions

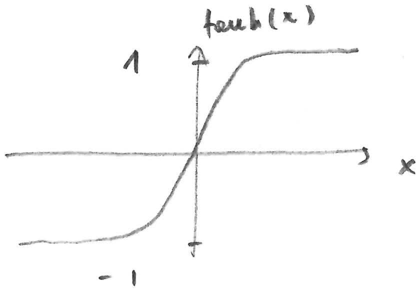
so far $\varphi(x) = \sigma(x) = \dfrac{1}{1+e^{-x}}$



:(sigmoid outputs are not centered (always positive)

:(if $|x| \gg 1$, $\sigma'(x) \approx \phi$ learning is slow!

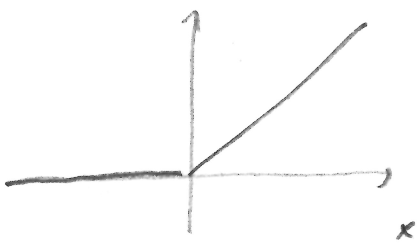
→ tanh activation function $\varphi(x) = \tanh(x)$



:) range $[-1, 1]$ and 0-centered

:(if $|x| \gg 1$, $\tanh'(x) \approx 0$

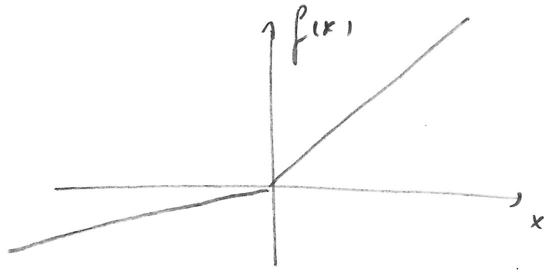
→ ReLU (rectified linear unit) activation function $f(x) = \max(0, x)$



:) does not saturate and converges faster than $\sigma(x)$ and $\tanh(x)$ in practice

:(what is the gradient for $x < 0$?

→ Leaky ReLU



$$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$$

→ Maxout generalizes ReLU and leaky ReLU

$$\max\left(W_1^T x + b_1 ,\ W_2^T x + b_2 \right)$$

:) linear regime, does not saturate, always well defined

:(doubles number of parameters to learn

"Maxout networks", Goodfellow, 2013

How to improve training?

→ L1 or L2 regularization

→ dropout

→ preprocess data (0-mean, 1-variance, dimensionality reduction via PCA)

→ Go deeper. Many hidden layers.

→ Convolutional neural networks which are not fully connected

→ Pre-train (in unsupervised way) each of the layers