Machine learning

MSc Social, Cognitive, and Affective Neuroscience SoSe 2020

Prof. Dr. Dirk Ostwald

(8) Neural networks

Model formulation

- Definitions and machine learning/deep learning/artificial intelligence jargon

Learning

- Cost function gradient descent

Backpropagation

- Cost function gradient evaluation

Universal approximation theorem

- The theory of function approximation by neural networks
- To be discussed in future iterations of the course

Model formulation

Learning

Backpropagation

Bibliographic remarks

Introductions to neural networks can be found in all standard machine learning books, such as Bishop (2006), Alpaydin (2014), Barber (2012), Duda et al. (2001), and Murphy (2012). Haykin (2009) is a classic reference on neural networks for engineers. Contemporary deep learning jargon and latest trends, but few mathematical and implementational details, are discussed in Goodfellow et al. (2017). Michael Nielsen's Online Book "Neural networks and deep learning" is a useful modern resource and not too confusing. A clear and useful reference on the backpropagation algorithm is Mishachev (2017).

Model formulation

Learning

Backpropagation

Overview

- Neural networks are parameterized multivariate, vector valued functions.
- Neural networks are serially concatenated linear and nonlinear functions.
- The (affine) linear functions are called *input weighting functions*.
- Input weighting functions parameters constitute the network's parameter set.
- The nonlinear functions are called *activation functions*.
- Activation functions do not have parameters.

Definition (Input weighting function, activation vector, weight matrix)

In the context of neural networks, affine-linear multivariate vector-valued functions of the form

$$\Phi : \mathbb{R}^n \to \mathbb{R}^m, a \mapsto \Phi(a) := Wa =: z \text{ with } W \in \mathbb{R}^{m \times n} \tag{1}$$

are called *input weighting functions*. The vector $a \in \mathbb{R}^n$ is called *activation vector*, the matrix $W \in \mathbb{R}^{m \times n}$ is called *weight matrix*, and the vector $z \in \mathbb{R}^m$ is called *weighted input*.

Definition (Coordinate-wise activation function, activation function)

In the context of neural networks, multivariate vector-valued functions of the form

$$\Sigma : \mathbb{R}^n \to \mathbb{R}^n, z \mapsto \Sigma(z) := \begin{pmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_n) \end{pmatrix} =: a, \tag{2}$$

with

$$\sigma : \mathbb{R} \to \mathbb{R}, z_i \mapsto \sigma(z_i) =: a_i \text{ for all } i = 1, ..., n \tag{3}$$

are called *component-wise activation functions*. The functions $\sigma : \mathbb{R} \to \mathbb{R}$ are called *activation functions*.

Definition (Homogeneous neural network)

A multivariate vector-valued function

$$f : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}, x \mapsto f(x) =: y \tag{4}$$

is called a *homogeneous k-layered neural network*, if f is of the form

$$f : \mathbb{R}^{n_0} \xrightarrow{\Phi^1} \mathbb{R}^{n_1} \xrightarrow{\Sigma^1} \mathbb{R}^{n_1} \xrightarrow{\Phi^2} \mathbb{R}^{n_2} \xrightarrow{\Sigma^2} \mathbb{R}^{n_2} \xrightarrow{\Phi^3} \dots$$
$$\dots \xrightarrow{\Phi^{k-1}} \mathbb{R}^{n_{k-1}} \xrightarrow{\Sigma^{k-1}} \mathbb{R}^{n_{k-1}} \xrightarrow{\Phi^k} \mathbb{R}^{n_k} \xrightarrow{\Sigma^k} \mathbb{R}^{n_k} \tag{5}$$

where

$$\Phi^l : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}, a^{l-1} \to \Phi^l\left(a^{l-1}\right) =: z^l \text{ for } l = 1, ..., k \tag{6}$$

are linear input weighting functions, and

$$\Sigma^l : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}, z^l \to \Sigma^l\left(z^l\right) =: a^l \text{ for } l = 1, ..., k \tag{7}$$

are coordinate-wise activation functions. For $x \in \mathbb{R}^{n_0}$, a homogeneous neural network takes on the value

$$f(x) := \Sigma^k\left(\Phi^k\left(\Sigma^{k-1}\left(\Phi^{k-1}\left(\Sigma^{k-2}\left(\cdots\left(\Sigma^1\left(\Phi^1\left(x\right)\right)\cdots\right)\right)\right)\right)\right)\right). \tag{8}$$

Remarks

- $a^l = \left(a_1^l, ..., a_{n_l}^l\right)^T \in \mathbb{R}^{n_l}$ is called *activation vector of layer* l.
- The $a_i^l \in \mathbb{R}$ for $i = 1, ..., n_l$ are called *neuron activations of layer* l.
- Layer $l = 0$ is called the *network input layer* and has dimension n_0.
- The activation vector of layer $l = 0$ is called *input* and is denoted $x := a^0$.
- Layer $l = k$ is called the *network output layer* and has dimension n_k.
- The activation vector of layer $l = k$ is called *output* and is denoted $y := a^k$.
- Layers $l = 1, ..., k-1$ are called *hidden layers*.
- Typically $n_l \geq n_0$ and $n_l > n_k$ for the hidden layers $1 \leq l \leq k-1$.

Remarks

- Let $W_{ij}^l \in \mathbb{R}$ denote the ijth entry in the lth "synaptic" weight matrix, i.e.,

$$W^l = \left(W_{ij}^l\right)_{1 \leq i \leq n_l, 1 \leq j \leq n_{l-1}} \in \mathbb{R}^{n_l \times n_{l-1}} \tag{9}$$

- W_{ij}^l is the "synaptic strength" between neuron i in layer l and neuron j in layer $l-1$.
- The weighted input ("post-synaptic potential") of neuron i in layer $l = 1, ..., k$ is

$$z_i^l = \sum_{j=1}^{n_{l-1}} W_{ij}^l a_j^{l-1}. \tag{10}$$

- The activation ("mean firing rate") of neuron i in layer $l = 1, ..., k$ is

$$a_i^l = \sigma\left(\sum_{j=1}^{n_{l-1}} W_{ij}^l a_j^{l-1}\right). \tag{11}$$

Definition (Activation function examples)

Commonly used activation functions and their derivatives are listed below.

Function name	Definition	Derivative
Standard logistic	$\sigma(z_i) := \frac{1}{1+\exp(z_i)}$	$\sigma'(z_i) = \frac{\exp(z_i)}{(1+\exp(z_i))^2}$
Hyperbolic tangent	$\sigma(z_i) := \tanh(z_i)$	$\sigma'(z_i) = 1 - \tanh^2(z_i)$
ReLU	$\sigma(z_i) := \max(0, z_i)$	$\sigma'(z_i) = \begin{cases} 0, & z_i < 0 \\ \emptyset, & z_i = 0 \\ 1, & z_i > 0 \end{cases}$
Leaky ReLU	$\sigma(z_i) := \begin{cases} 0.01 z_i, & z_i \leq 0 \\ z_i, & z_i > 0 \end{cases}$	$\sigma'(z_i) = \begin{cases} 0.01, & z_i \leq 0 \\ 1, & z_i > 0 \end{cases}$

Remarks

- ReLU is the acronym for "rectified linear unit".

Example (Structure of a homogeneous neural network)

$k = 3, n_0 = 2, n_1 = 3, n_2 = 3$ and $n_3 = 1$.

$x =: a_0$

$$a^0 = \begin{pmatrix} a_1^0 \\ a_2^0 \end{pmatrix} \quad W^1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \\ w_{31}^1 & w_{32}^1 \end{pmatrix} \quad z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} \quad \Sigma^1 = \begin{pmatrix} \sigma\left(z_1^1\right) \\ \sigma\left(z_2^1\right) \\ \sigma\left(z_3^1\right) \end{pmatrix} = a^1$$

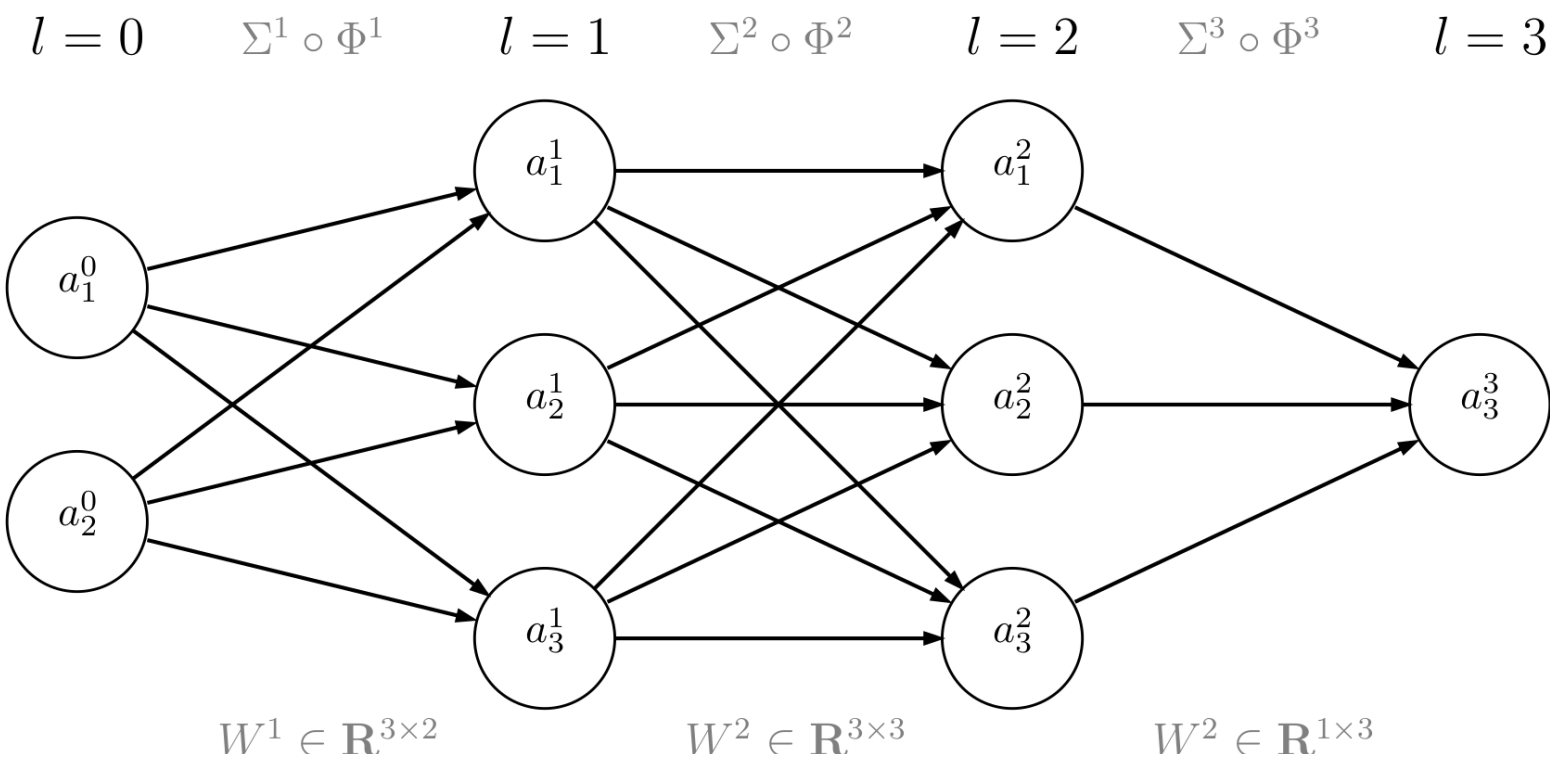
$$a^1 = \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} \quad W^2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 \\ w_{31}^2 & w_{32}^2 & w_{33}^2 \end{pmatrix} \quad z^2 = \begin{pmatrix} z_1^2 \\ z_2^2 \\ z_3^2 \end{pmatrix} \quad \Sigma^2 = \begin{pmatrix} \sigma\left(z_1^2\right) \\ \sigma\left(z_2^2\right) \\ \sigma\left(z_3^2\right) \end{pmatrix} = a^2$$

$$a^2 = \begin{pmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{pmatrix} \quad W^3 = \begin{pmatrix} w_{11}^3 & w_{12}^3 & w_{13}^3 \end{pmatrix} \quad z^3 = \begin{pmatrix} z_1^3 \end{pmatrix} \quad \Sigma^3 = \begin{pmatrix} \sigma\left(z_1^3\right) \end{pmatrix} = a^3$$

$a^3 =: y$

Example (Structure of a homogeneous network)

$k = 3, n_0 = 2, n_1 = 3, n_2 = 3$ and $n_3 = 1$.



Theorem (Affine neural network)

An affine neural network can be obtained from a homogeneous neural network by

(1) augmenting the input vector by a 1, such that $x = (x_1, ..., x_{n_0-1}, 1)^T$,

(2) augmenting the weight matrices W^l by a row of form $(0, ..., 0, 1)$ for all $l < k$,

(3) setting $\sigma\left(z^l_{n_l}\right) := \mathrm{id}\left(z^l_{n_l}\right)$ for all $l < k$.

Remarks

- The entries of the last columns of W^l, bar the last entry, are called *biases*.
- The W^l row $(0, ..., 0, 1)$ serves to reproduce the activation vector augmentation.
- The identity function evaluates to $\mathrm{id}\left(z^l_{n_l}\right) = \mathrm{id}(1) = 1$ for $l < k$.
- The activation of neuron $i, i = 1, ..., n_l - 1$ in layer $l = 1, ..., k-1$ is

$$a^l_i = \sigma\left(\sum_{j=1}^{n_{l-1}-1} W^l_{ij} a^{l-1}_j + b^l_i\right) \text{ with bias } b^l_i := W^l_{i,n_{l-1}}. \tag{12}$$

- Instead of a proof, we consider an example.

Example (Homogeneous and affine neural network structures)

Homogeneous neural network for $k = 2, n_0 = 2, n_1 = 3, n_2 = 1$, and $a^0 := x$

$$a^0 = \begin{pmatrix} a_1^0 \\ a_2^0 \end{pmatrix} \qquad W^1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \\ w_{31}^1 & w_{32}^1 \end{pmatrix} \qquad z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \end{pmatrix} \qquad \Sigma^1 = \begin{pmatrix} \sigma\left(z_1^1\right) \\ \sigma\left(z_2^1\right) \\ \sigma\left(z_3^1\right) \end{pmatrix} = a^1$$

$$a^1 = \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} \qquad W^2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \end{pmatrix} \qquad z^2 = \begin{pmatrix} z_1^2 \end{pmatrix} \qquad \Sigma^2 = \begin{pmatrix} \sigma\left(z_1^2\right) \end{pmatrix} = y$$

Affine neural network for $k = 2, n_0 = 3, n_1 = 4, n_2 = 1$, and $a^0 := x$

$$a^0 = \begin{pmatrix} a_1^0 \\ a_2^0 \\ 1 \end{pmatrix} \qquad W^1 = \begin{pmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ 0 & 0 & 1 \end{pmatrix} \qquad z^1 = \begin{pmatrix} z_1^1 \\ z_2^1 \\ z_3^1 \\ 1 \end{pmatrix} \qquad \Sigma^1 = \begin{pmatrix} \sigma\left(z_1\right) \\ \sigma\left(z_2\right) \\ \sigma\left(z_3\right) \\ \mathsf{id}(1) \end{pmatrix} = a^1$$

$$a^1 = \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ 1 \end{pmatrix} \qquad W^2 = \begin{pmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 \end{pmatrix} \qquad z^2 = \begin{pmatrix} z_1^2 \end{pmatrix} \qquad \Sigma^2 = \begin{pmatrix} \sigma\left(z_1^2\right) \end{pmatrix} = y$$

Definition (Hadamard operator form of network function values)

Let

$$\Sigma : \mathbb{R}^n \to \mathbb{R}^n, z \mapsto \Sigma(z) := \Sigma \left(\begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \right) = \begin{pmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_n) \end{pmatrix} \tag{13}$$

denote a component-wise activation function. Then we write

$$\Sigma \circ z := \Sigma(z) \tag{14}$$

and call this the *Hadamard operator form* of a component-wise activation function. With the Hadamard operator form of the component-wise activation functions and the standard matrix product, the value of a network function with argument x can be written as

$$f(x) = \Sigma^L \circ W^L \cdot \Sigma^{k-1} \circ W^{k-1} \cdot \Sigma^{L-2} \cdots \Sigma^2 \circ W^2 \cdot \Sigma^1 \circ W^1 \cdot x. \tag{15}$$

Model formulation

Learning

Backpropagation

Learning

Overview

- Training refers to adjusting the neural network's weight parameters.
- Good parameters minimize the deviation between predicted and training data output.
- This deviation is formalized in terms of a *cost function*.
- Various cost functions are in common use.
- Gradient descent is used to minimize the cost with respect to the parameters.
- In machine learning lingo, neural network training is a form of *supervised learning*.
- From a statistics perspective, this is a parameter estimation problem.

Definition (Neural network training set)

A *neural network training set* is a set of vector pairs

$$\mathcal{D} := \{(x^{(i)}, y^{(i)})\}_{i=1}^{n} \in \mathbb{R}^{n_0 \times n} \times \mathbb{R}^{n_k \times n}, \tag{16}$$

where $x^{(i)} \in \mathbb{R}^{n_0}$ is referred to as *feature vector* and $y^{(i)} \in \mathbb{R}^{n_k}$ is referred to as *target vector*. Typical target vector formats include

$y^{(i)} \in \{0, 1\}$	Binary classification
$y^{(i)} \in \{0, 1\}^{n_k}, \sum_{i=1}^{n_k} y_i = 1, n_k > 1$	n_k-fold classification with "one-hot-encoding"
$y^{(i)} \in \mathbb{R}^{n_k}, n_k > 1$	Regression problems

For example, in fMRI MVPA multiclass classification

- n_0 is the number of voxels, $x^{(i)} \in \mathbb{R}^{n_0}$ is a voxel activation pattern,
- the $y^{(i)} \in \{0, 1\}^{n_k}, \sum_{i=1}^{n} y_i = 1$ for $n_k > 1$ label experimental conditions, and
- n is the number of voxel activation pattern/condition observations.

Definition (Neural network training)

Neural network training is the process of adapting the vector of weight matrices

$$\mathcal{W} := \left\{\text{vec}\left(W^l\right)\right\}_{l=1}^{k} \in \mathbb{R}^p, W^l \in \mathbb{R}^{n_l \times n_{l-1}} \text{ for } l = 1, ..., k \text{ and } p := \sum_{l=1}^{k} n_{l-1} n_l \quad (17)$$

with the aim of minimizing a deviation criterion between the weight-adapted neural networks final layer activation $f\left(x^{(i)}\right) \in \mathbb{R}^{n_k}$ and the associated value of the target vector $y^{(i)} \in \mathbb{R}^{n_k}$ across all training exemplars $\left(x^{(i)}, y^{(i)}\right), i = 1, ..., n$ in the training set $\mathcal{D}$. To emphasize the dependence of a neural network's input-specific output value on its parameter vector, we write

$$f_{\mathcal{W}}(x) := f(x) \in \mathbb{R}^{n_k} \text{ for } x \in \mathbb{R}^{n_0} \quad (18)$$

in the following.

Remarks

- A neural network has p parameters.
- The "criterion of deviation" is formalized in terms of a cost function.

Definition (Neural network cost functions)

An *additive neural network cost function* is a function of the form

$$C_{\mathcal{D}} : \mathbb{R}^p \to \mathbb{R}, \;\; \mathcal{W} \mapsto C_{\mathcal{D}}\left(\mathcal{W}\right) := \frac{1}{n} \sum_{i=1}^{n} c\left(f_{\mathcal{W}}\left(x^{(i)}\right), y^{(i)}\right), \tag{19}$$

where

$$c : \mathbb{R}^{n_0} \times \mathbb{R}^{n_k} \to \mathbb{R}, (f_{\mathcal{W}}(x), y) \mapsto c\left(f_{\mathcal{W}}(x), y\right) \tag{20}$$

denotes a *training exemplar cost function*. With $a := f_{\mathcal{W}}(x) = a^k$, we may equivalently write

$$c : \mathbb{R}^{n_0} \times \mathbb{R}^{n_k} \to \mathbb{R}, (a, y) \mapsto c\left(a, y\right) \tag{21}$$

leaving the dependence of the training exemplar cost function on the input vector x and the neural network's parameter set $\mathcal{W}$ implicit. Note that we write the vector of partial derivatives of a training exemplar cost function with respect to the elements $a_i, i = 1, ..., n_k$ evaluated for input $x \in \mathbb{R}^{n_0}$ and output $y \in \mathbb{R}^{n_k}$ as

$$\nabla_a c\left(a, y\right) = \left(\frac{\partial}{\partial a_1} c(a, y), ..., \frac{\partial}{\partial a_{n_k}} c(a, y)\right)^T \in \mathbb{R}^{n_k}. \tag{22}$$

Definition (Additive cost function examples)

For a training set $\mathcal{D} := \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ and with $a^{(i)} := f_{\mathcal{W}}\left(x^{(i)}\right)$ for $i = 1, ..., n$, exemplary additive cost functions are

Quadratic cost $$C_{\mathcal{D}}(\mathcal{W}) := \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} ||a^{(i)} - y^{(i)}||$$

Exponential cost $$C_{\mathcal{D}}(\mathcal{W}) := \frac{1}{n} \sum_{i=1}^{n} \exp\left(-\sum_{j=1}^{n_k} \left(a_j^{(i)} - y_j^{(i)}\right)^2\right)$$

Cross entropy cost $$C_{\mathcal{D}}(\mathcal{W}) := -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n_k} y_j^{(i)} \ln a_j^{(i)} + \left(1 - y_j^{(i)}\right) \ln\left(1 - a_j^{(i)}\right),$$

Definition (Neural network gradient descent)

Let $\mathcal{D}$ denote a neural network training data set comprising n training exemplars, let $f_{\mathcal{W}}$ denote a k-layered neural network with parameter vector $\mathcal{W} \in \mathbb{R}^p$, and let $C_{\mathcal{D}}$ denote an additive neural network cost function with associated training exemplar cost function c. Then a gradient descent algorithm for the optimization of the neural network's parameter vector is given by

Initialization

Set $\mathcal{W}^{(0)} \in \mathbb{R}^p$ and $\alpha > 0$ appropriately.

Iterations

For $j = 1, 2, ...$ until convergence set

$$\mathcal{W}^{(j)} := \mathcal{W}^{(j-1)} - \alpha \sum_{i=1}^{n} \nabla c \left(f_{\mathcal{W}^{(j-1)}} \left(x^{(i)} \right), y^{(i)} \right), \tag{23}$$

where ∇c denotes the gradient of the training exemplar cost function with respect to the neural network parameter vector $\mathcal{W} \in \mathbb{R}^p$.

Remark

- $\mathcal{W}^{(j)}$ is adapted in the average gradient direction over training exemplars.

Model formulation

Learning

Backpropagation

Motivation

- Neural network gradient descent parameter learning requires $\nabla c\left(f_{\mathcal{W}}(x), y\right)$.
- $\nabla c\left(f_{\mathcal{W}}(x), y\right)$ comprises all partial derivatives

$$\frac{\partial}{\partial W_{ij}^{l}} c\left(f_{\mathcal{W}}(x), y\right) \text{ for } i = 1, ..., n_l, j = 1, ..., n_{l-1}, l = 1, ..., k. \tag{24}$$

- A naive approach is to estimate $\nabla c\left(f_{\mathcal{W}}(x), y\right)$ by the numerical derivatives

$$\frac{\partial}{\partial W_{ij}^{l}} c\left(f_{\mathcal{W}}(x), y\right) \approx \frac{1}{\epsilon}\left(c\left(f_{\tilde{\mathcal{W}}}(x), y\right) - c\left(f_{\mathcal{W}}(x), y\right)\right) \text{ with } \tilde{W} := \mathcal{W} + \epsilon e_{ij}^{l} \tag{25}$$

  for a small $\epsilon > 0$ and where $e_{ij}^{l} \in \mathbb{R}^p$ is a vector of 0s bar a 1 at location i, j, l.
- This naive approach requires $n_{l-1} \cdot n_l \cdot l + 1$ evaluations of c and hence of $f_{\mathcal{W}}(x)$.
- A single evaluation of c for a training exemplar (x, y) is called a *forward pass*.
- *Backpropagation* is an alternative algorithm for computing $\nabla c\left(f_{\mathcal{W}}(x), y\right)$.
- Backpropagation requires only a single forward pass and a *backward pass*.
- In the following, we introduce the "backward pass" recursion

Theorem (Backpropagation recursion for neural networks)

Let $f_{\mathcal{W}}$ denote a k-layered neural network, let c denote a single training exemplar cost function, and let

$$\tilde{\Sigma}^l := \left(\sigma', ..., \sigma'\right)^T \in \mathbb{R}^{n_l} \tag{26}$$

denote the derivative of the neural network's lth layer's component-wise activation function. Then the partial derivatives of c with respect to the neural network parameters W^l for $l = k, k-1, ..., 1$ can be computed according to the following *backpropagation recursion*:

Initialization

Set $W^{k+1} := 1$ and $\delta^{k+1} := \nabla_a c\left(a^k, y\right)$.

Iterations

For $l = k, k-1, k-2, ..., 1$, set

$$\delta^l = \left(\left(W^{l+1}\right)^T \delta^{l+1}\right) \circ \tilde{\Sigma}^l\left(z^l\right) \text{ and } \frac{\partial}{\partial W^l} c\left(f_{\mathcal{W}}(x), y\right) := \delta^l \Sigma^{l-1}\left(z^{l-1}\right)^T, \tag{27}$$

where $\Sigma^0\left(z^0\right) := x$.

Proof

We prove the validity of the backpropagation recursion by induction with respect to the number of layers j of a neural network. To this end, we first validate the backpropagation recursion directly for the case of $k := 3$ (base case). We then assume the validity of the backpropagation recursion for some k and show that it is also valid for a neural network with an additional layer, i.e., for $k + 1$ (inductive step). Throughout, we make repeated use of the chain rule of differentiation, i.e.,

$$D(f \circ g)(x) = Df(g(x))Dg(x), \tag{28}$$

and we make repeated use of the matrix derivative

$$\frac{\partial}{\partial A} AB = B^T. \tag{29}$$

Base case validation for $k := 3$

We consider the case of a 3-layered neural network, i.e., a multivariate real-valued function of the form

$$f_{\mathcal{W}}(x) := \Sigma^3 \left(W^3 \Sigma^2 \left(W^2 \Sigma^1 \left(W^1 x \right) \right) \right) \tag{30}$$

and its partial derivatives with respect to W^3, W^2, and W^1. In the following, we will first evaluate the respective partial derivatives directly and then evaluate them using the backpropagation recursion to validate their equivalence.

Proof (cont.)

We first evaluate the partial derivative $\frac{\partial}{\partial W^3} c(f_{\mathcal{W}}(x), y)$ directly. To this end, we first note that the only appearance of W^3 is in the following formulation of the cost function

$$c\left(f_{\mathcal{W}}(x), y\right) = c\left(\Sigma^3\left(W^3 \Sigma^2\left(z^2\right)\right), y\right). \tag{31}$$

We then have

$$\begin{aligned}
\frac{\partial}{\partial W^3} c\left(f_{\mathcal{W}}(x), y\right) &= \frac{\partial}{\partial W^3} c\left(\Sigma^3\left(W^3 \Sigma^2\left(z^2\right)\right), y\right) \\
&= \nabla_a c\left(a^3, y\right) \frac{\partial}{\partial W^3}\left(\Sigma^3\left(W^3 \Sigma^2\left(z^2\right)\right), y\right) \\
&= \nabla c\left(a^3, y\right) \tilde{\Sigma}^3\left(W^3 \Sigma^2\left(z^2\right)\right) \frac{\partial}{\partial W^3}\left(W^3 \Sigma^2\left(z^2\right)\right) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right)\left(\Sigma^2\left(z^2\right)\right)^T
\end{aligned} \tag{32}$$

We next evaluate $\frac{\partial}{\partial W^3} c(f_{\mathcal{W}}(x), y)$ by means of the backpropagation recursion. To this end, we first note that for $l = 3$ and with $W^4 = 1$

$$\begin{aligned}
\delta^4 &= \nabla_a c\left(a^3, y\right) \\
\delta^3 &= \delta^4 \left(W^4\right)^T \tilde{\Sigma}^3\left(z^3\right) = \nabla_a c\left(a^3, y\right) \cdot 1 \cdot \tilde{\Sigma}^3\left(z^3\right) = \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right)
\end{aligned}$$

Proof (cont.)

For the partial derivative with $l = 3$, we thus have

$$\frac{\partial}{\partial W^3} c\left(f_{\mathcal{W}}(x), y\right) = \delta^3 \Sigma^2\left(z^2\right) \tag{33}$$

$$= \nabla c\left(a^3\right) \tilde{\Sigma}^3\left(z^3\right)\left(\Sigma^2\left(z^2\right)\right)^T \tag{34}$$

which corresponds to the partial derivative as evaluated directly above.

We next evaluate the partial derivative $\frac{\partial}{\partial W^2} c(f_{\mathcal{W}}(x), y)$ directly. To this end, we first note that the only appearance of W^2 is in the following formulation of the cost function

$$c\left(f_{\mathcal{W}}(x), y\right) = c\left(\Sigma^3\left(W^3 \Sigma^2\left(W^2 \Sigma^1\left(z^1\right)\right)\right)\right) \tag{35}$$

With the chain rule of differentiation, we then have

$$\begin{aligned}
\frac{\partial}{\partial W^2} c\left(f_{\mathcal{W}}(x), y\right) &= \frac{\partial}{\partial W^2} c\left(\Sigma^3\left(W^3 \Sigma^2\left(W^2 \Sigma^1\left(z^1\right)\right)\right), y\right) \\
&= \nabla_a c\left(a^3, y\right) \frac{\partial}{\partial W^2}\left(\Sigma^3\left(W^3 \Sigma^2\left(W^2 \Sigma^1\left(z^1\right)\right)\right)\right) \qquad (36) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) \frac{\partial}{\partial W^2}\left(W^3 \Sigma^2\left(W^2 \Sigma^1\left(z^1\right)\right)\right)
\end{aligned}$$

Proof (cont.)

$$
\begin{aligned}
\ldots &= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \frac{\partial}{\partial W^2}\left(\Sigma^2\left(W^2 \Sigma^1\left(z^1\right)\right)\right) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \tilde{\Sigma}^2\left(z^2\right) \frac{\partial}{\partial W^2}\left(W^2 \Sigma^1\left(z^1\right)\right) \qquad (37) \\
&= \nabla_a c\left(a^3, y\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right)\left(\Sigma^1\left(z^1\right)\right)^T
\end{aligned}
$$

We next evaluate $\frac{\partial}{\partial W^2} c(f_{\mathcal{W}}(x), y)$ by means of the backpropagation recursion. To this end, we first note that for $l = 2$ and with the results for $l = 3$ above

$$
\begin{aligned}
\delta^3 &= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) \\
\delta^2 &= \delta^3 W^3 \tilde{\Sigma}^2\left(z^2\right) &&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \tilde{\Sigma}^2\left(z^2\right)
\end{aligned}
$$

For the partial derivative with $l = 2$, we thus have

$$
\begin{aligned}
\frac{\partial}{\partial W^2} c\left(f_{\mathcal{W}}(x), y\right) &= \delta^2\left(\Sigma^1\left(z^1\right)\right)^T \qquad (38) \\
&= \nabla_a c\left(a^3, y\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right)\left(\Sigma^1\left(z^1\right)\right)^T, \qquad (39)
\end{aligned}
$$

which corresponds to the partial derivative as evaluated directly above.

Proof (cont.)

We next evaluate the partial derivative $\frac{\partial}{\partial W^1} c(f_{\mathcal{W}}(x), y)$ directly. To this end, we first note that the only appearance of the parameter W^1 is in the following formulation of the cost function

$$c(f_{\mathcal{W}}(x), y) = c\left(\Sigma^3\left(W^3\Sigma^2\left(W^2\Sigma^1\left(W^1 x\right)\right)\right)\right). \tag{40}$$

With the chain rule of differentiation, we then have

$$\begin{aligned}
&\frac{\partial}{\partial W^1} c(f_{\mathcal{W}}(x), y) \\
&= \frac{\partial}{\partial W^1} c\left(\Sigma^3\left(W^3\Sigma^2\left(W^2\Sigma^1\left(W^1 x\right)\right)\right), y\right) \\
&= \nabla_a c\left(a^3, y\right) \frac{\partial}{\partial W^1}\left(\Sigma^3\left(W^3\Sigma^2\left(W^2\Sigma^1\left(W^1 x\right)\right)\right)\right) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \frac{\partial}{\partial W^1}\left(\Sigma^2\left(W^2\Sigma^1\left(W^1 x\right)\right)\right) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \tilde{\Sigma}^2\left(z^2\right) W^2 \frac{\partial}{\partial W^1}\left(\Sigma^1\left(W^1 x\right)\right) \\
&= \nabla_a c\left(a^3, y\right) \tilde{\Sigma}^3\left(z^3\right) W^3 \tilde{\Sigma}^2\left(z^2\right) W^2 \tilde{\Sigma}^1\left(z^1\right) \frac{\partial}{\partial W^1}\left(W^1 x\right) \\
&= \nabla_a c\left(a^3, y\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right) W^2 \cdot \tilde{\Sigma}^1\left(z^1\right) x^T.
\end{aligned} \tag{41}$$

Proof (cont.)

We next evaluate $\frac{\partial}{\partial W^1} c\left(f_{\mathcal{W}}(x), y\right)$ by means of the backpropagation recursion. To this end, we first note that for $l = 1$ and with the results for $l = 2$ above

$$\begin{aligned}
\delta^2 &= \nabla_a c\left(a^3, y\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right) \\
\delta^1 &= \delta^2 W^2 \cdot \tilde{\Sigma}^1\left(z^1\right) \\
&= \nabla_a c\left(a^3\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right) W^2 \cdot \tilde{\Sigma}^1\left(z^1\right)
\end{aligned}$$

For the partial derivative with $l = 1$ and with $\Sigma^0(z^0) := x$ we thus have

$$\begin{aligned}
\frac{\partial}{\partial W^1} c(f_{\mathcal{W}}(x), y) &= \delta^1 \left(\Sigma^0\left(z^0\right)\right)^T \\
&= \nabla_a c\left(a^3, y\right) \cdot \tilde{\Sigma}^3\left(z^3\right) W^3 \cdot \tilde{\Sigma}^2\left(z^2\right) W^2 \cdot \tilde{\Sigma}^1\left(z^1\right) \left(\Sigma^0\left(z^0\right)\right)^T
\end{aligned}$$

which corresponds to the partial derivative as evaluated directly above. This completes the validation of the base case for $k = 3$.

Proof (cont.)

Induction step: $k + 1$

To be continued ...

Model formulation

- Definitions and machine learning/deep learning/artificial intelligence jargon

Learning

- Cost function gradient descent

Backpropagation

- Cost function gradient evaluation

Universal approximation theorem

- The theory of function approximation by neural networks
- To be discussed in future iterations of the course

References

Alpaydin, E. (2014). *Introduction to Machine Learning*.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York.

Duda, R., Hart, P., and Stork, D. (2001). *Pattern Classification*. Wiley.

Goodfellow, I., Bengio, Y., and Courville, A. (2017). *Deep Learning*. The Mit Press, Cambridge, Massachusetts.

Haykin, S. S. (2009). *Neural Networks and Learning Machines*. Prentice Hall, New York, 3rd ed edition.

Mishachev, N. M. (2017). Backpropagation in matrix notation. *arXiv:1707.02746 [cs]*.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA.