QBUS6850 Week 2

Neural Networks 2
Training a Neural Network

Dr Stephen Tierney

The University of Sydney Business School

Recommended Reading

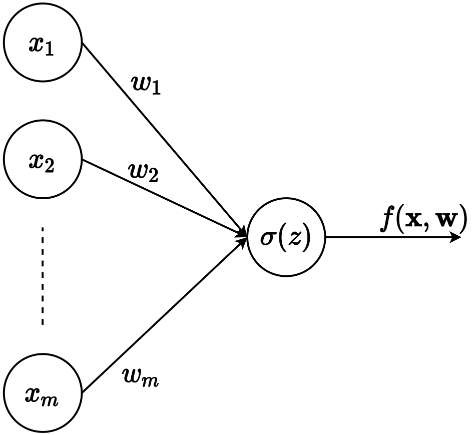
- Chapter 11.4-11.5, The Elements of Statistical Learning, Hastie et al.
- Chapter 5.3, Pattern Recognition and Machine Learning, Christopher Bishop
- Chapter 6.5, Deep Learning, Goodfellow et al.

For a summary of matrix calculus please see the wikipedia page: `https://en.wikipedia.org/wiki/Matrix_calculus`

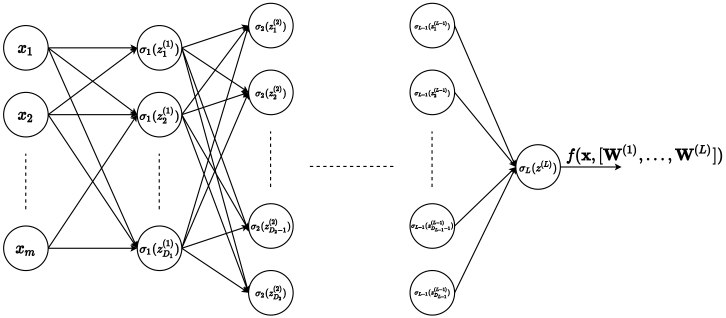
The Plan

- Week 1: Neural Network Introduction
- **Week 2: Training a Neural Network**
- Week 3: Refining the Training Process
- Week 4: Network Design

What You Know - Artificial Neurons



What You Know - Artificial Neural Networks



x_1
x_2
x_m
$\sigma_1(z_1^{(1)})$
$\sigma_1(z_2^{(1)})$
$\sigma_1(z_{D_1}^{(1)})$
$\sigma_2(z_1^{(2)})$
$\sigma_2(z_2^{(2)})$
$\sigma_2(z_{D_2-1}^{(2)})$
$\sigma_2(z_{D_2}^{(2)})$
$\sigma_{L-1}(z_1^{(L-1)})$
$\sigma_{L-1}(z_2^{(L-1)})$
$\sigma_{L-1}(z_{D_{L-1}-1}^{(L-1)})$
$\sigma_{L-1}(z_{D_{L-1}}^{(L-1)})$
$\sigma_L(z^{(L)})$
$f(\mathbf{x}, [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}])$

What is “training” a neural network?

So far our neurons and networks have all been fixed.

Training a network refers to **selecting optimal parameters** for a network.

Parameters vs Hyper-Parameters

Parameters

- Weights between neurons

Hyper-parameters

- Number of layers
- Dimension (number of neurons) in each layer
- Activation function at each layer
- Loss Function

Hyper-parameters

Hyper-parameters describe the type of problem that our network can solve.

For example:

- to solve multi-class classification we need multiple output neurons
- to estimate a probability we will need something a final activation function with range $(0, 1)$ (e.g. sigmoid)

Parameters

The parameters or weights adjust the importance of features and their interactions with respect to the target variable or label, based on the training data at hand.

Parameters

A network trained on one dataset may not perform well on another dataset, even though the task is the same.

This is the same as most models. For example the parameters of a linear regression needs to be trained for each new dataset.

Estimating Parameters

For today we will assume that the hyper-parameters of our network are fixed.

This means we don't worry about

- Number of layers
- Dimension (number of neurons) in each layer
- Activation function at each layer
- Loss Function

Estimating Parameters

Our objective is always to minimise our chosen loss function L over a set of training data.

Expressed mathematically:

$$\min \sum_{i=1}^{N} L(y_i, \hat{y}_i))$$

where y_i is the true label and $\hat{y}_i$ is the network's prediction for data point i.

Estimating Parameters

Expanding using definitions from last week:

$$\begin{aligned}\min \sum_{i=1}^{N} L(y_i, \hat{y}_i)) &= \min \sum_{i=1}^{N} L(y_i, \mathbf{A}^{(L)}) \\ &= \min_{\mathbf{W}^{(1)},\ldots,\mathbf{W}^{(L)}} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i, [\mathbf{W}^{(1,\ldots,L)}]))\end{aligned}$$

where $f(\mathbf{x}_i, [\mathbf{W}^{(1,\ldots,L)}])$ is the prediction from the network for a data point.

In other words **find the weight matrices, $\mathbf{W}^{(l)}$, that minimise the loss over the training data**.

Gradient Descent

If the loss function is convex we can use gradient descent:

1. Initialise θ and step size α
2. Repeat until convergence
    2.1 Update $\theta_k = \theta_{k-1} - \alpha \frac{\partial L}{\partial \theta_{k-1}}$

Gradient Descent

For a neural network there is a matrix of parameters at each layer.

Therefore we need to update them all at the same iteration i.e. we need to compute

$$\mathbf{W}_k^{(l)} = \mathbf{W}_{k-1}^{(l)} - \alpha \frac{\partial L}{\partial \mathbf{W}_{k-1}^{(l)}}$$

for $l = 1, \ldots, L$

Helpful Definitions

Composite Functions

A composite function nests one function inside another.

For example F is a composite function of f and g

$$F(x) = f(g(x))$$

Composite Functions

Neural networks are composite functions.

Composite Functions

Recall the forward propagation formula

$$\begin{aligned} f(\mathbf{x}, [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}]) &= \mathbf{a}^{(L)} \\ &= \sigma_L\left(\mathbf{z}^{(L)}\right) \\ &= \sigma_L\left(\mathbf{a}^{(L-1)}\mathbf{W}^{(L)}\right) \end{aligned}$$

If we set $f = \sigma_L$ and $g(\mathbf{a}^{(L-1)}) = \mathbf{a}^{(L-1)}\mathbf{W}^{(L)}$ then we have

$$f(\mathbf{x}, [\mathbf{W}^{(1,\ldots,L)}]) = f\left(g(\mathbf{a}^{(L-1)})\right)$$

and this extends to other layers of the network since $\mathbf{a}^{(L-1)}$ is also the result of a composition at the previous layer.

Chain Rule

If F is a composite function then the derivative of F is

$$F'(x) = f'(g(x)) \cdot g'(x).$$

In this lecture we will use Leibniz's notation where we set $F = f(u)$ and $u = g(x)$

$$\frac{dF}{dx} = \frac{dF}{du} \cdot \frac{du}{dx}$$

Chain Rule

The chain rule tells us how to calculate the partial derivatives at each layer.

As before if we set $F = \sigma_L(\mathbf{z})$ and $\mathbf{z} = g(\mathbf{a}^{(L-1)}) = \mathbf{a}^{(L-1)}\mathbf{W}^{(L)}$ then

$$\frac{\partial F}{\partial \mathbf{W}^{(L)}} = \frac{\partial F}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(L)}}$$

If we expand we get

$$\frac{\partial F}{\partial \mathbf{W}^{(L)}} = \frac{\partial \sigma_L(\mathbf{z})}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{a}^{(L-1)}\mathbf{W}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

Training a Network

Backpropagation

Backpropagation is an efficient method of calculating the partial derivatives for each layer.

The idea is that the derivative of parameters at early layers depend on the derivative at later layers.

Therefore we should compute the derivatives in a layer wise manner, starting at the last layer and working back.

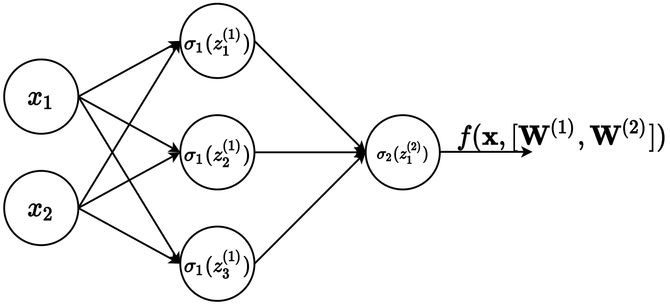
Training Algorithm

1. Initialise all weight matrices
2. Repeat until stopping criteria satisfied
   - 2.1 Forward propagate
   - 2.2 Backpropagate to compute partial derivatives
   - 2.3 Update weights for all layers

$$\mathbf{W}_k^{(l)} = \mathbf{W}_{k-1}^{(l)} - \alpha \frac{\partial L}{\partial \mathbf{W}_{k-1}^{(l)}}$$

Example

Let's train the network shown below for a regression task.



Example

Settings:

1. Activation Functions

$$\sigma_1(z) = \sigma_2(z) = z$$

2. Loss function

$$L = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Example

Weight matrices/vectors are of the form

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & w_{1,3}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & w_{2,3}^{(1)} \end{bmatrix}$$

$$\mathbf{w}^{(2)} = \begin{bmatrix} w_{1}^{(2)} \\ w_{2}^{(2)} \\ w_{3}^{(2)} \end{bmatrix}$$

Example

Training Data

x_1	x_2	y
1	0.875	1
0.25	1	0.4
0.75	0.25	0.6

We won't end up using this in the example, but it is useful to keep in mind!

Example

In this example we will derive the formulas for backpropagation.

If we were to train the network we would evaluate the formulas using the current values of our weight matrices.

Initalisation

We need to fill the weight matrices with initial values.

This is typically done with **random** values.

Example - Forward Propagation

You know how to do this from last week!

Example - Forward Propagation

Why do we need to do a forward propagation?

Because the partial derivative formulas will involve

$$\hat{y}_i = f(\mathbf{x}_i, [\mathbf{W}^{(1,\dots,L)}])$$

which is the output of the network for data point i at the current iteration.

Specifically they will be used to calculate the error between our predictions and the targets e.g.

$$e_i = \hat{y}_i - y_i$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

Starting from the last (right-most) layer we need to calculate

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = \begin{bmatrix} \frac{\partial L}{\partial w_1^{(2)}} \\ \frac{\partial L}{\partial w_2^{(2)}} \\ \frac{\partial L}{\partial w_3^{(2)}} \end{bmatrix}$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

First use the sum rule of differentiation

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{w}^{(2)}} &= \frac{\partial \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{\partial \mathbf{w}^{(2)}} \\ &= \sum_{i=1}^{N} \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \mathbf{w}^{(2)}}\end{aligned}$$

Since each term in the sum is independent we can find the solution for each data point and sum later. So we will work on the following **single training point** case

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \mathbf{w}^{(2)}}$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

Applying the chain rule

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}^{(2)}} &= \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \mathbf{w}^{(2)}} \\
&= \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} \\
&= (\hat{y}_i - y_i) \frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}}
\end{aligned}$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

Since

$$\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} = \frac{\partial \sigma_2(z^{(2)})}{\partial \mathbf{w}^{(2)}}$$

We can apply the chain rule

$$\begin{aligned}\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} &= \frac{\partial \sigma_2(z^{(2)})}{\partial \mathbf{w}^{(2)}} \\ &= \frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial \mathbf{w}^{(2)}}\end{aligned}$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

Since σ_2 is the identity activation function

$$\frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} = \frac{\partial z^{(2)}}{\partial z^{(2)}} = 1$$

and since $z^{(2)} = \mathbf{a}^{(1)}\mathbf{w}^{(2)}$

$$\frac{\partial z^{(2)}}{\partial \mathbf{w}^{(2)}} = \frac{\partial \mathbf{a}^{(1)}\mathbf{w}^{(2)}}{\partial \mathbf{w}^{(2)}} = \mathbf{a}^{(1)}$$

Example - Backpropagation

w.r.t $\mathbf{w}^{(2)}$

Putting it all together we get

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = (\hat{y}_i - y_i)\mathbf{a}^{(1)}$$

in the single data point case.

Since we are using “numerator layout“ for the derivatives we need to transpose for dimensional consistency

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = (\hat{y}_i - y_i)\mathbf{a}^{(1)\top}$$

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

Continuing to the next layer

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \begin{bmatrix} \frac{\partial L}{\partial w^{(1)}_{1,1}} & \frac{\partial L}{\partial w^{(1)}_{1,2}} & \frac{\partial L}{\partial w^{(1)}_{1,3}} \\ \frac{\partial L}{\partial w^{(1)}_{2,1}} & \frac{\partial L}{\partial w^{(1)}_{2,2}} & \frac{\partial L}{\partial w^{(1)}_{2,3}} \end{bmatrix}$$

Again we will work on the scalar (single training point) case first

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \mathbf{W}^{(1)}}$$

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

Continue to apply the chain rule

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{W}^{(1)}} &= \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \mathbf{W}^{(1)}} \\
&= \frac{\partial \frac{1}{2}(\hat{y}_i - y_i)^2}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{y}_i - y_i) \frac{\hat{y}_i}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{y}_i - y_i) \frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}} \\
&= (\hat{y}_i - y_i) \frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}}
\end{aligned}$$

Since earlier we discovered that $\frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} = 1$.

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

This leaves us to work on

$$\frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}}$$

which we solve by the chain rule (surprise!)

$$\begin{aligned}
\frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}} &= \frac{\partial z^{(2)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{W}^{(1)}} \\
&= \frac{\partial z^{(2)}}{\partial \mathbf{a}^{(1)}} \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}
\end{aligned}$$

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

Partial derivatives are defined for column-wise vectors. Therefore for consistency we transpose all denominator row vectors. The results can be transposed for the dimension consistent solution.

$$\frac{\partial z^{(2)}}{\partial \mathbf{a}^{(1)\top}} = \frac{\partial \mathbf{a}^{(1)}\mathbf{w}^{(2)}}{\partial \mathbf{a}^{(1)\top}} = \mathbf{w}^{(2)\top}$$

$$\frac{\partial z^{(2)}}{\partial \mathbf{a}^{(1)}} = \mathbf{w}^{(2)}$$

$$\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} = \frac{\partial \sigma_1(\mathbf{z}^{(1)})}{\partial \mathbf{z}^{(1)}} = 1$$

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

To deal with the last term we expand

$$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{x}\mathbf{W}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

which appears tricky because we don't have a nice definition for vector-by-matrix derivatives.

However we can use the fact that the multiplication is independent column-wise and therefore decomposable as

$$\mathbf{x}\mathbf{W}^{(1)} = \sum_{p=1}^{D_1} \mathbb{1}_p \odot \mathbf{x}\mathbf{W}^{(1)}_{:,p}$$

where $\mathbf{W}^{(1)}_{:,p}$ is the pth column of $\mathbf{W}^{(1)}$, $\mathbb{1}_p$ is an indicator vector with 1 at index p and 0 everywhere else and $\odot$ is the element-wise product.

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

$$\begin{aligned}
\frac{\partial \mathbf{x}\mathbf{W}^{(1)\top}}{\partial \mathbf{W}^{(1)}} &= \sum_{p=1}^{D_1} \frac{\partial \mathbb{1}_p \odot \mathbf{x}\mathbf{W}^{(1)}_{:,p}}{\partial \mathbf{W}^{(1)}_{:,p}} \\
&= \sum_{p=1}^{D_1} \mathbb{1}_p \odot \frac{\partial \mathbf{x}\mathbf{W}^{(1)}_{:,p}}{\partial \mathbf{W}^{(1)}_{:,p}} \\
&= \sum_{p=1}^{D_1} \mathbb{1}_p \odot \mathbf{x} \\
&= \mathbf{x}
\end{aligned}$$

Example - Backpropagation

w.r.t $\mathbf{W}^{(1)}$

Putting everything together

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}^{(1)}} &= (\hat{y}_i - y_i)\frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}} \\ &= e_i \mathbf{w}^{(2)} \mathbf{x}\end{aligned}$$

Transposing for dimensional consistency

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{W}^{(1)}} &= (\hat{y}_i - y_i)\frac{\partial z^{(2)}}{\partial \mathbf{W}^{(1)}} \\ &= e_i \mathbf{x}^\top \mathbf{w}^{(2)\top}\end{aligned}$$

Backpropagation - Summary

The chain rule allows us to compute derivatives at each layer.

The derivatives at earlier layers depend on the derivatives of later layers.

Therefore to avoid wasted computation, we should proceed in a layer wise fashion from back to front hence the name: **backpropagation**.

Classification

In the classification case we would use a sigmoid activation at the final layer

$$a^{(2)} = \sigma_2(z^{(2)})$$
$$\sigma_2(z) = \frac{1}{1 + e^{-z}}$$

and our loss function is the cross entropy

$$L = -\sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

How does this affect our partial derivative calculations?

Classification

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = -\frac{\partial y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)}{\partial \mathbf{w}^{(2)}}$$

Applying the chain rule

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}^{(2)}} &= -\frac{\partial y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)}{\partial \mathbf{w}^{(2)}} \\
&= -\frac{\partial y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} \\
&= -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right)\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} \\
&= -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right)\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}}
\end{aligned}$$

Classification

$$\begin{aligned}\frac{\partial \hat{y}_i}{\partial \mathbf{w}^{(2)}} &= \frac{\partial \hat{y}_i}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial \mathbf{w}^{(2)}} \\ &= \frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial \mathbf{w}^{(2)}}\end{aligned}$$

Classification

We are using the sigmoid activation function

$$\sigma_2(z) = \frac{1}{1+e^{-z}}$$
$$\frac{d\sigma_2(z)}{dz} = \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma_2(z)(1-\sigma_2(z))$$

$$\begin{aligned}\frac{\partial \sigma_2(z^{(2)})}{\partial z^{(2)}} &= \sigma_2(z^{(2)})(1-\sigma_2(z^{(2)})) \\ &= a^{(2)}(1-a^{(2)})\end{aligned}$$

and since $z^{(2)} = \mathbf{a}^{(1)}\mathbf{w}^{(2)}$

$$\begin{aligned}\frac{\partial z^{(2)}}{\partial \mathbf{w}^{(2)}} &= \frac{\partial \mathbf{a}^{(1)}\mathbf{w}^{(2)}}{\partial \mathbf{w}^{(2)}} \\ &= \mathbf{a}^{(1)}\end{aligned}$$

Classification

Joining everything back together

$$\frac{\partial L}{\partial \mathbf{w}^{(2)}} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1-y_i}{1-\hat{y}_i}\right) a^{(2)}(1-a^{(2)})\mathbf{a}^{(1)}$$

Noting that $\hat{y}_i = a^{(2)}$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{w}^{(2)}} &= \left(-y_i + y_i a^{(2)} + a^{(2)} - y_i a^{(2)}\right)\mathbf{a}^{(1)} \\ &= \left(a^{(2)} - y_i\right)\mathbf{a}^{(1)} \\ &= (\hat{y}_i - y_i)\,\mathbf{a}^{(1)}\end{aligned}$$

Which is the same as in the case of regression using an identity activation!

Wrapping Up

Summary

Today you learnt:

- backpropagation
- how to apply the chain rule to calculate partial derivatives at any layer
- how to train a neural network given fixed hyper-parameters and a dataset

Summary

Remaining issues:

- Number of layers
- Dimension (number of neurons) in each layer
- Activation function at each layer
- Loss Function

What's Next?

- Week 1: Neural Network Introduction
- Week 2: Training a Neural Network
- **Week 3: Refining the Training Process**
- Week 4: Network Design