LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Deep Learning

Chapter 3: Basic Backpropagation 1

Bernd Bischl

Department of Statistics – LMU Munich

WS 2021/2022

BACKPROPAGATION: BASIC IDEA

In DL, we aim to optimize the empirical risk by gradient descent

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

where $\boldsymbol{\theta}$ are the weights (and biases) of the network.
Training of NNs happens in 2 consecutive steps, for one observation $\mathbf{x}$:

1. **Forward pass**: The information of the inputs flow through the model to produce a prediction. Based on that, we compute the empirical risk. We covered that.
2. **Backward pass** / **Backpropagation**: The information of the error that happened in the prediction of $\mathbf{x}$ flows backwards through the model, we calculate the error contribution of each weight to update weights by the negative gradient.

This is simply gradient descent in disguise (for one observation).

WEIGHT UPDATE RULE

- Backpropagation can then be used to compute the gradient of $L\left(y, f(\mathbf{x} \mid \boldsymbol{\theta})\right)$ in an **extremely** efficient way.
- The weight update per iteration for one $\mathbf{x}$ with learning rate α, is

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \cdot \nabla_{\theta} L\left(y, f\left(\mathbf{x} \mid \boldsymbol{\theta}^{[t]}\right)\right)$$

- We will see that, at its core, backpropagation is just a clever implementation of the chain rule. Nothing more!
- We could now sum up these gradients for all $\mathbf{x}^{(i)}$ from $\left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$ to compute the gradient over the complete training set, to perform a full GD step. But as we want to arrive at stochastic gradient descent, we stick for now with updates for a single $\mathbf{x}$.

BACKPROPAGATION EXAMPLE

- Let us recall the XOR example, but this time with randomly initialized weights.
- As activations in both the hidden and output layers we apply the **sigmoidal logistic function**.
- To perform one forward and one backward pass we feed our neural network with example $\mathbf{x} = (1, 0)^T$ (positive sample).
- We will optimize the model using the squared error between the binary 0-1 labels and the predicted probabilities instead of the Bernoulli loss. This is a bit unusual but computations become simpler for this instructive example.
- Then we compute the backward pass and apply backpropagation to update the weights.
- Finally we evaluate the model with our updated weights.

Note: We will only show rounded decimals.

BACKPROPAGATION EXAMPLE

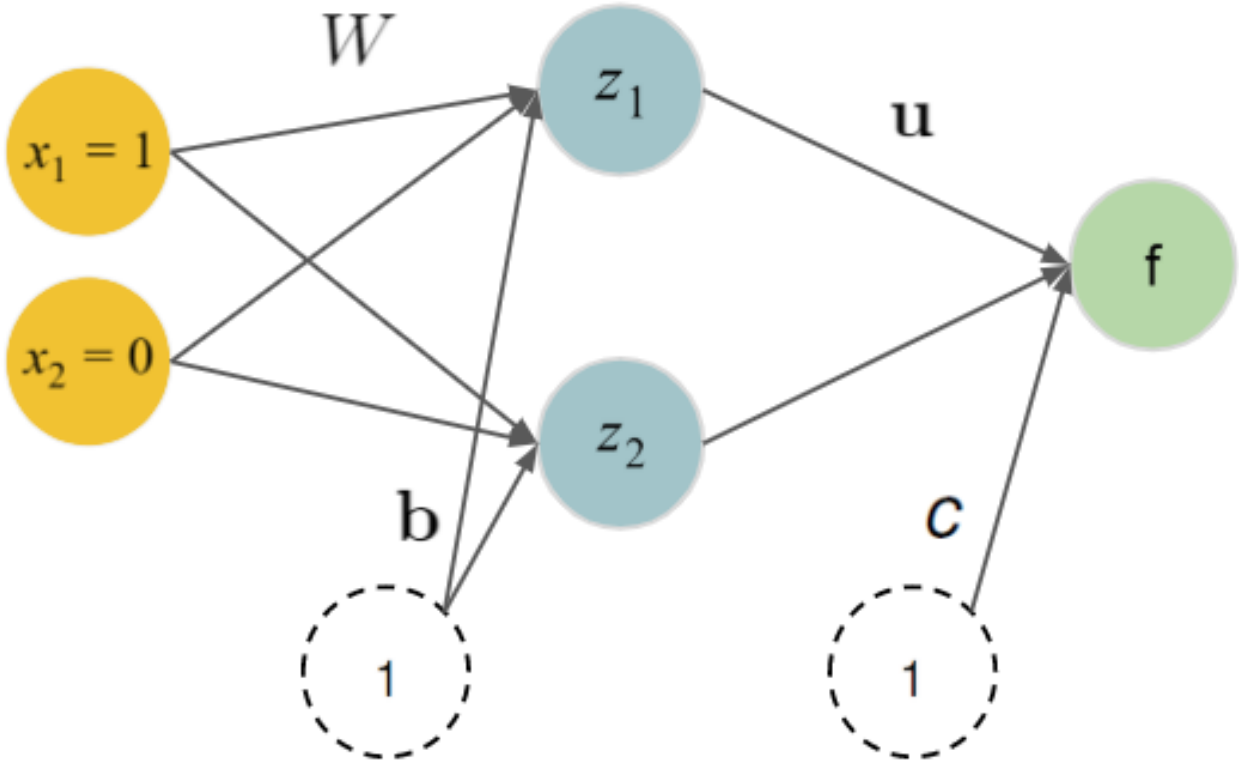
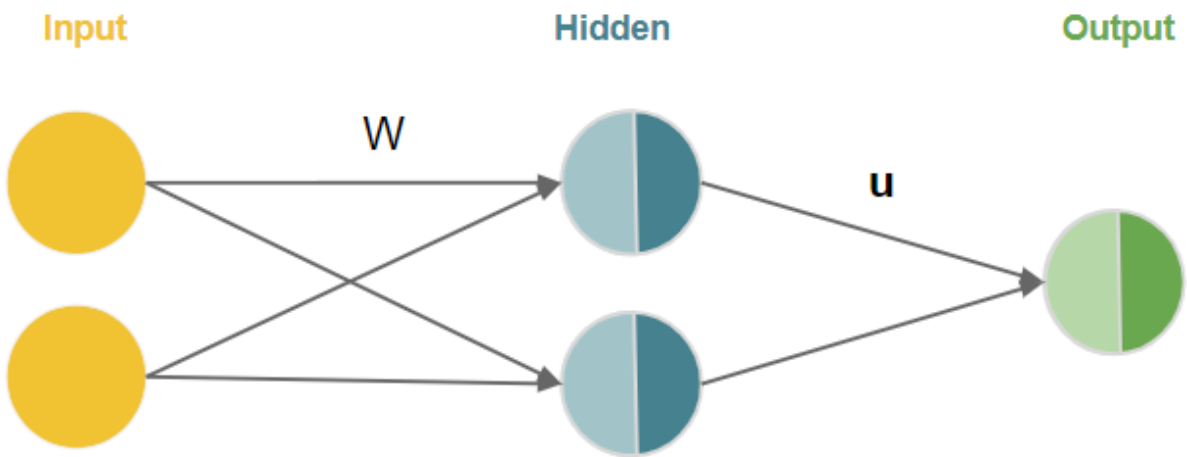


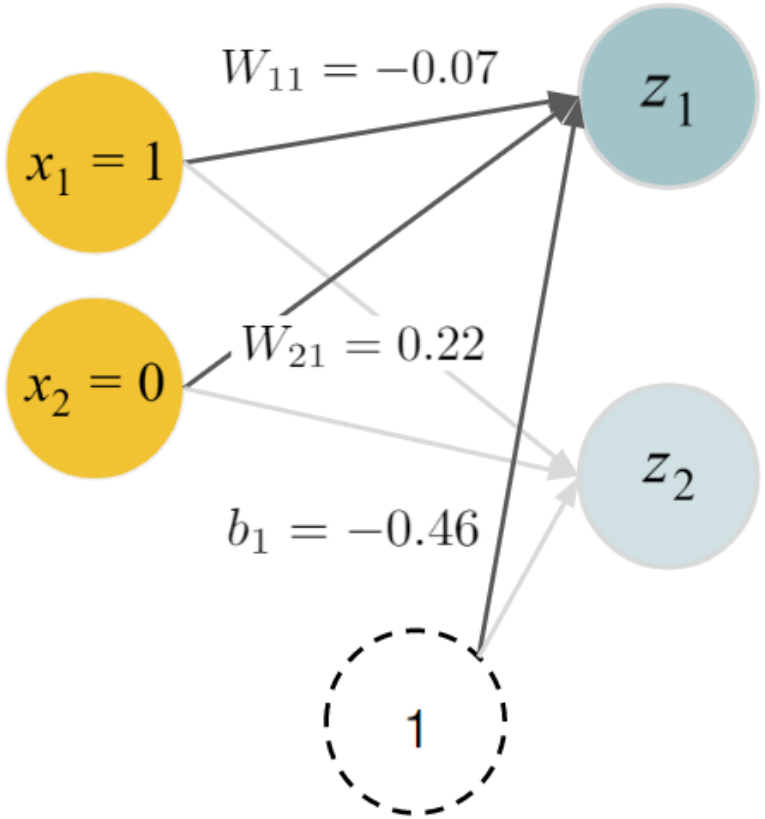
Figure: A neural network with two neurons in the hidden layer. W and **b** are the weights and biases of the hidden layer. **u** and c are the weights and bias of the output layer. Based on the values for the weights (and biases), we will perform one forward and one backward pass.

BACKPROP EXAMPLE: FORWARD PASS

- We will divide the forward pass into four steps:
  - the inputs of z_i: $z_{i,in}$
  - the activations of z_i: $z_{i,out}$
  - the input of f: f_{in}
  - and finally the activation of f: f_{out}



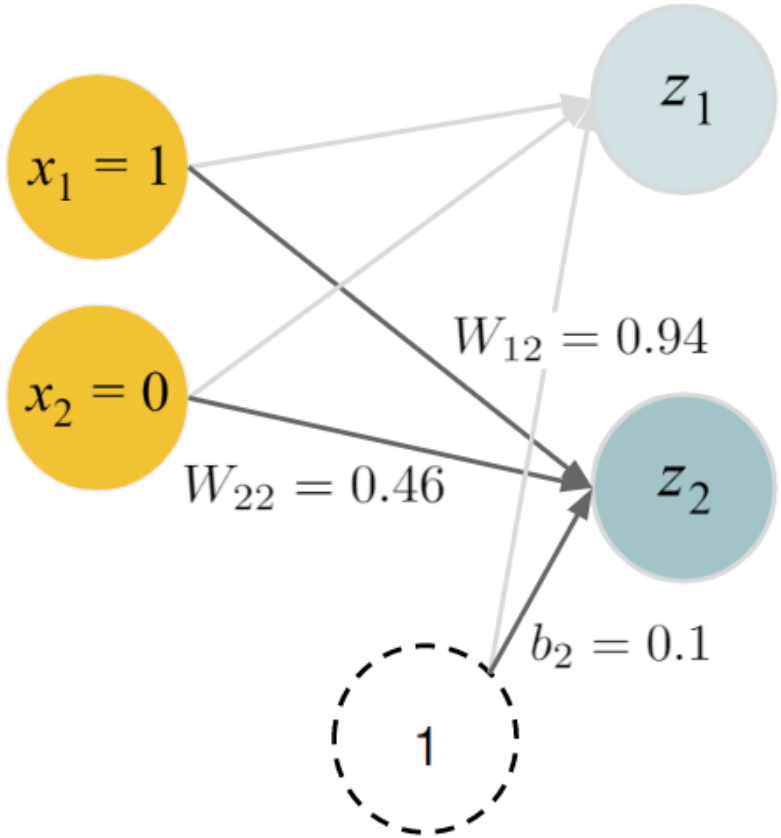
BACKPROP EXAMPLE: FORWARD PASS



$$z_{1,in} = \mathbf{W}_1^T \mathbf{x} + b_1 = 1 \cdot (-0.07) + 0 \cdot 0.22 + 1 \cdot (-0.46) = -0.53$$

$$z_{1,out} = \sigma(z_{1,in}) = \frac{1}{1 + \exp(-(-0.53))} = 0.3705$$

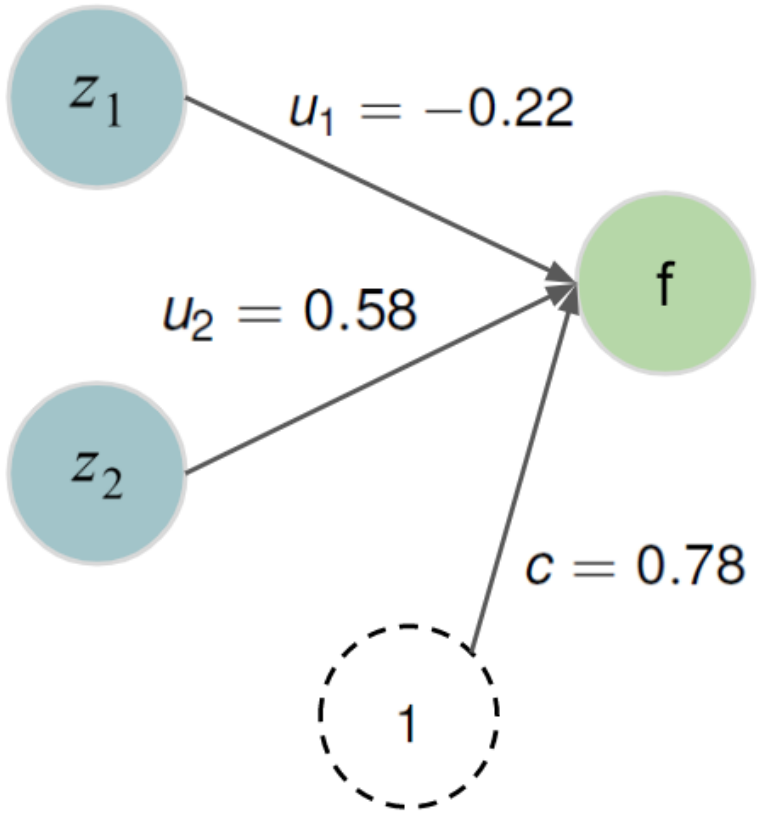
BACKPROP EXAMPLE: FORWARD PASS



$$z_{2,in} = \mathbf{W}_2^T \mathbf{x} + b_2 = 1 \cdot 0.94 + 0 \cdot 0.46 + 1 \cdot 0.1 = 1.04$$

$$z_{2,out} = \sigma(z_{2,in}) = \frac{1}{1 + \exp(-1.04)} = 0.7389$$

BACKPROP EXAMPLE: FORWARD PASS



$$f_{in} = \boldsymbol{u}^T \boldsymbol{z} + c = 0.3705 \cdot (-0.22) + 0.7389 \cdot 0.58 + 1 \cdot 0.78 = 1.1122$$

$$f_{out} = \tau(f_{in}) = \frac{1}{1 + \exp(-1.1122)} = 0.7525$$

BACKPROP EXAMPLE: FORWARD PASS

- The forward pass of our neural network predicted a value of

$$f_{out} = 0.7525$$

- Now, we compare the prediction $f_{out} = 0.7525$ and the true label $y = 1$ using the L2-loss:

$$\begin{aligned} L(y, f(\mathbf{x})) &= \frac{1}{2}(y - f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right))^2 = \frac{1}{2}(y - f_{out})^2 \\ &= \frac{1}{2}(1 - 0.7525)^2 = 0.0306 \end{aligned}$$

- The calculation of the gradient is performed backwards (starting from the output layer), so that results can be reused.

BACKPROP EXAMPLE: WEIGHT UPDATES

- We will see that the main ingredients to perform the backward pass are:
  - to reuse the results of the forward pass (here: $z_{i,in}$, $z_{i,out}$, f_{in}, f_{out})
  - to reuse the intermediate results during the backward pass due to the chain rule
  - to calculate the derivative of some activation functions and some affine functions
- This is demonstrated by continuing the example above.

BACKPROP EXAMPLE: WEIGHT UPDATES

- Assume we would like to know how much and in which direction a change in u_1 affects the total error. We recursively apply the chain rule and compute:

$$\frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} = \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial u_1}$$

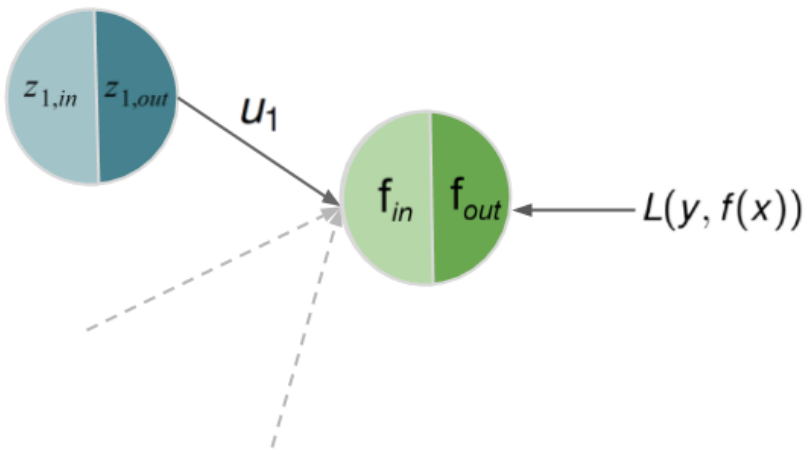


Figure: Snippet from our neural network showing the backward path to compute the gradient with respect to weight u_1.

BACKPROP EXAMPLE: WEIGHT UPDATES

- 1st step (backwards): We know f_{out} from the forward pass.

$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} &= \frac{d}{\partial f_{out}} \frac{1}{2}(y - f_{out})^2 = -\underbrace{(y - f_{out})}_{\hat{=}\text{residual}} \\ &= -(1 - 0.7525) = -0.2475\end{aligned}$$

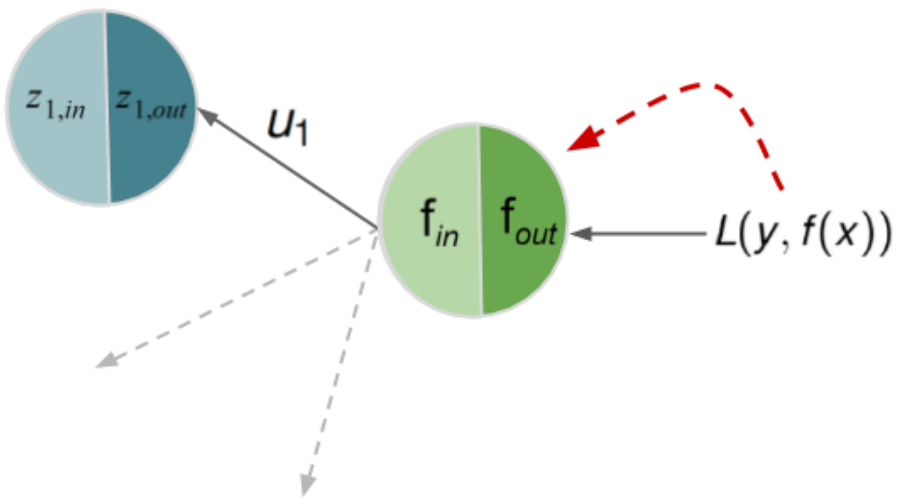


Figure: The first term of our chain rule $\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- 2nd step (backwards). $f_{out} = \sigma(f_{in})$ and we apply the rule for σ' (see Chapter 1-3). We already know f_{in} from the forward pass.

$$\begin{aligned} \frac{\partial f_{out}}{\partial f_{in}} &= \sigma(f_{in}) \cdot (1 - \sigma(f_{in})) \\ &= 0.7525 \cdot (1 - 0.7525) = 0.1862 \end{aligned}$$

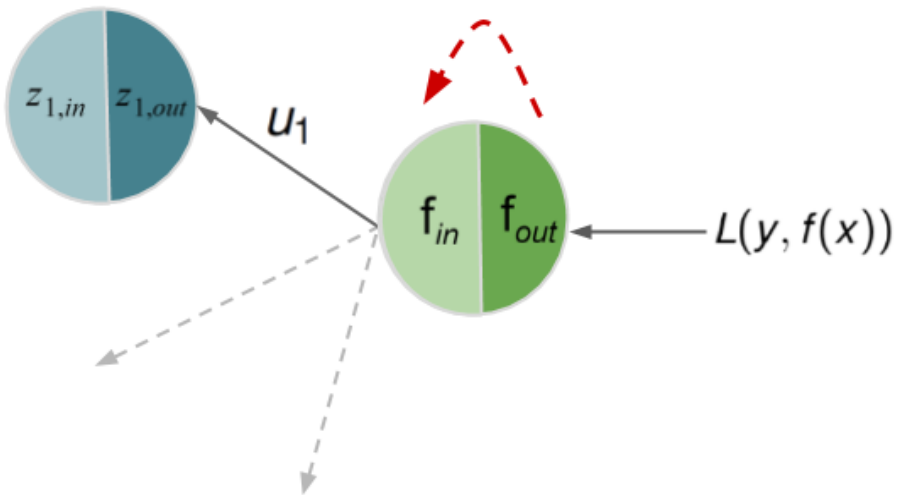


Figure: The second term of our chain rule $\frac{\partial f_{out}}{\partial f_{in}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- 3rd step (backwards). We know $z_{1,out}$ from the forward pass.

$$\frac{\partial f_{in}}{\partial u_1} = \frac{\partial(u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out} + c \cdot 1)}{\partial u_1} = z_{1,out} = 0.3705$$

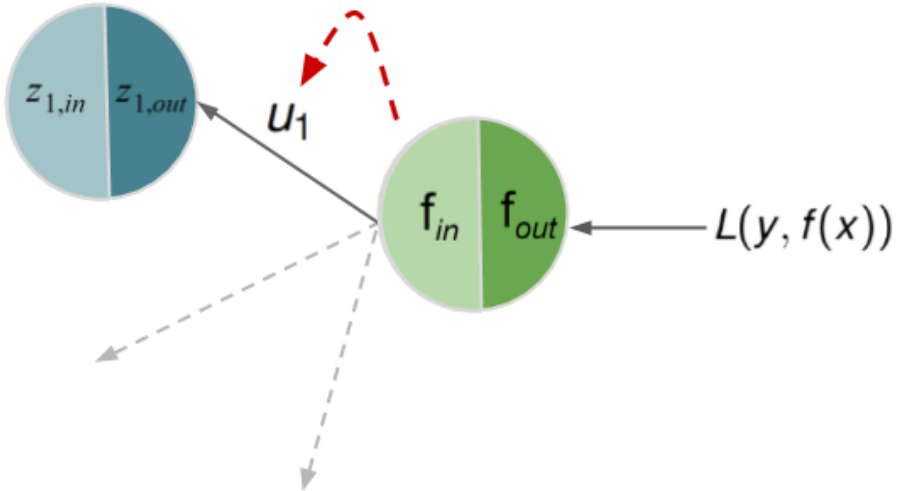


Figure: The third term of our chain rule $\frac{\partial f_{in}}{\partial u_1}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- Finally we are able to plug all three parts together and obtain:

$$\begin{aligned}\frac{\partial L(y, f(\mathbf{x}))}{\partial u_1} &= \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial u_1} \\ &= -0.2475 \cdot 0.1862 \cdot 0.3705 = -0.0171\end{aligned}$$

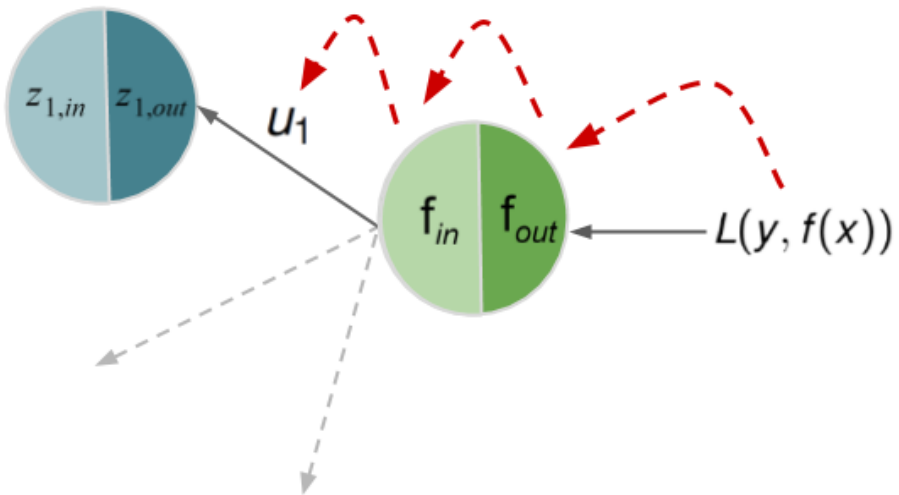


Figure: All three terms of our chain rule $\frac{\partial L(y,f(\mathbf{x}))}{\partial u_1} = \frac{\partial L(y,f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial u_1}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- Consider a learning rate of $\alpha = 0.5$. Then we obtain:

$$\begin{aligned} u_1^{[new]} &= u_1^{[old]} - \alpha \cdot \frac{\partial L\left(y, f(\mathbf{x})\right)}{\partial u_1} \\ &= -0.22 - 0.5 \cdot (-0.0171) \\ &= -0.2115 \end{aligned}$$

BACKPROP EXAMPLE: WEIGHT UPDATES

- We now also want to do the same for W_{11}. We have to compute:

$$\frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} = \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial z_{1,out}} \cdot \frac{\partial z_{1,out}}{\partial z_{1,in}} \cdot \frac{\partial z_{1,in}}{\partial W_{11}}$$

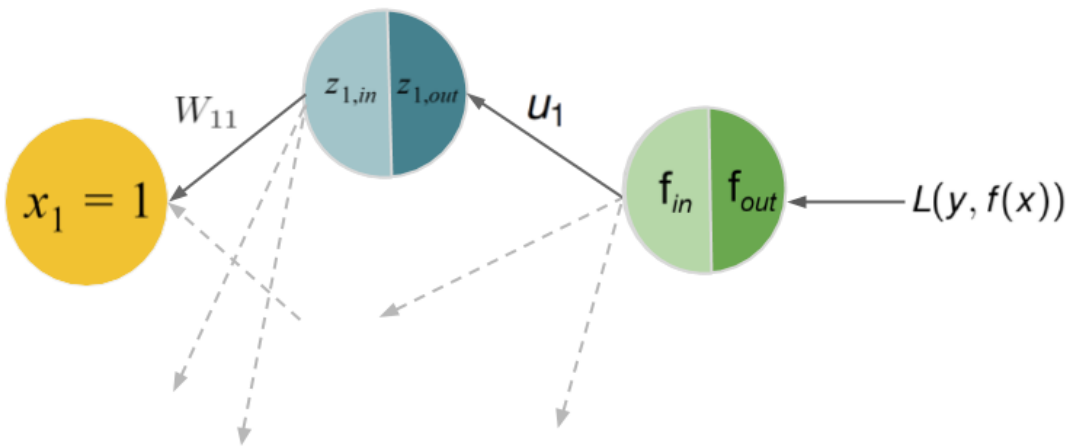


Figure: Snippet from our neural network showing the backward path to compute the gradient with respect to weight W_{11}.

BACKPROP EXAMPLE: WEIGHT UPDATES

- We already know $\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}}$ and $\frac{\partial f_{out}}{\partial f_{in}}$ from the backward pass before for updating u_1.

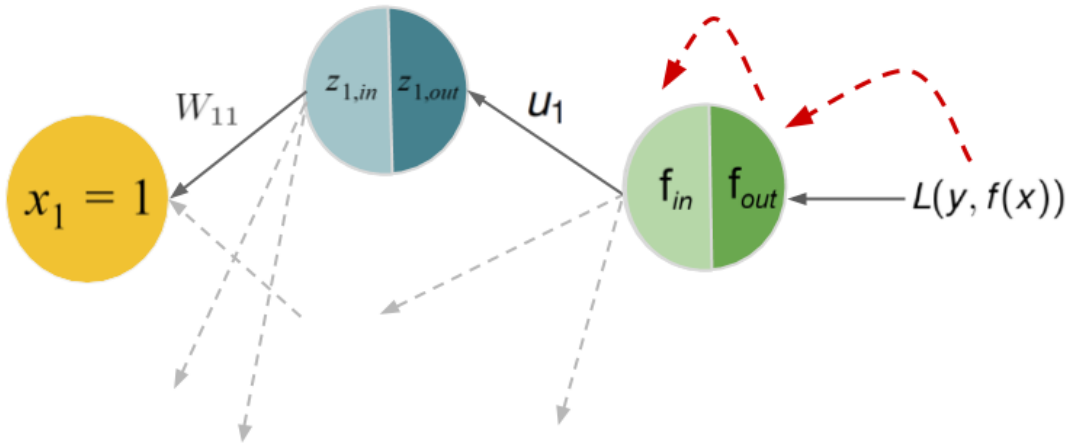


Figure: The first and second term of our chain rule $\frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}}$ and $\frac{\partial f_{out}}{\partial f_{in}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- With $f_{in} = u_1 \cdot z_{1,out} + u_2 \cdot z_{2,out} + c \cdot 1$ we can compute:

$$\frac{\partial f_{in}}{\partial z_{1,out}} = u_1 = -0.22$$

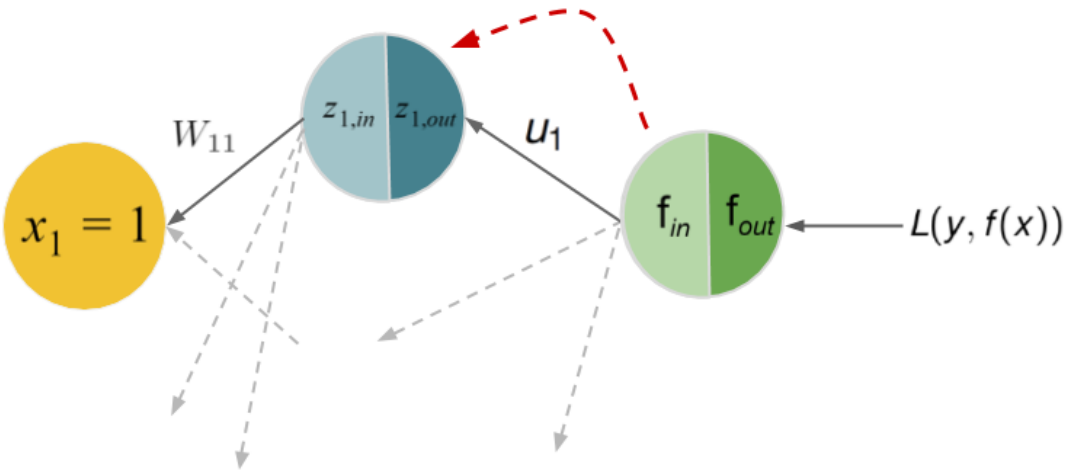


Figure: The third term of our chain rule $\frac{\partial f_{in}}{\partial z_{1,out}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- Next, we need

$$\begin{aligned} \frac{\partial z_{1,out}}{\partial z_{1,in}} &= \sigma(z_{1,in}) \cdot (1 - \sigma(z_{1,in})) \\ &= 0.3705 \cdot (1 - 0.3705) = 0.2332 \end{aligned}$$

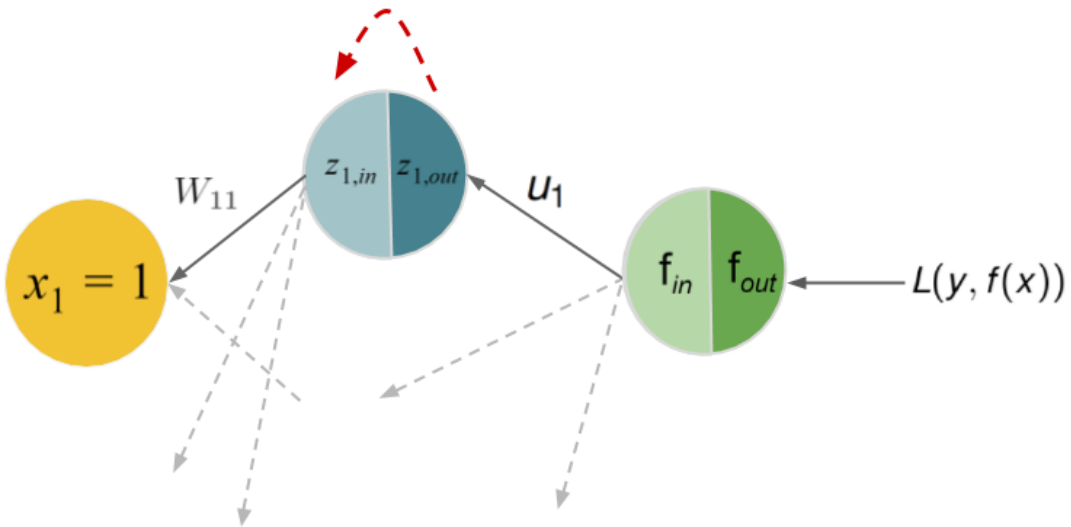


Figure: The fourth term of our chain rule $\frac{\partial z_{1,out}}{\partial z_{1,in}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- With $z_{1,in} = x_1 \cdot W_{11} + x_2 \cdot W_{21} + b_1 \cdot 1$ we can compute the last component:

$$\frac{\partial z_{1,in}}{\partial W_{11}} = x_1 = 1$$

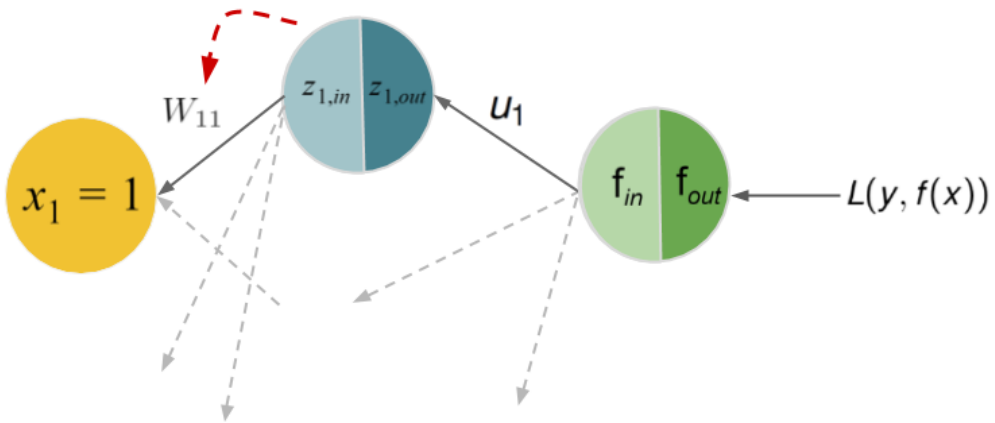


Figure: The fifth term of our chain rule $\frac{\partial z_{1,in}}{\partial W_{11}}$

BACKPROP EXAMPLE: WEIGHT UPDATES

- Plugging all five components together yields us:

$$\begin{aligned}
\frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} &= \frac{\partial L(y, f(\mathbf{x}))}{\partial f_{out}} \cdot \frac{\partial f_{out}}{\partial f_{in}} \cdot \frac{\partial f_{in}}{\partial z_{1,out}} \cdot \frac{\partial z_{1,out}}{\partial z_{1,in}} \cdot \frac{\partial z_{1,in}}{\partial W_{11}} \\
&= (-0.2475) \cdot 0.1862 \cdot (-0.22) \cdot 0.2332 \cdot 1 \\
&= 0.0024
\end{aligned}$$

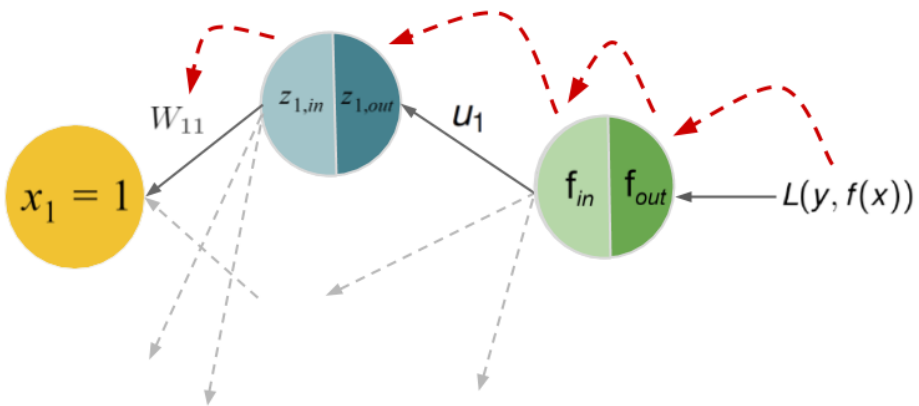


Figure: All five terms of our chain rule

BACKPROP EXAMPLE: WEIGHT UPDATES

- Consider the same learning rate of $\alpha = 0.5$. Then we obtain:

$$\begin{aligned} W_{11}^{[new]} &= W_{11}^{[old]} - \alpha \cdot \frac{\partial L(y, f(\mathbf{x}))}{\partial W_{11}} \\ &= -0.07 - 0.5 \cdot 0.0024 = -0.0712 \end{aligned}$$

- We would now like to check how the performance has improved. Our updated weights are:

$$W = \begin{pmatrix} -0.0712 & 0.9426 \\ 0.22 & 0.46 \end{pmatrix}, b = \begin{pmatrix} -0.4612 \\ 0.1026 \end{pmatrix},$$

$$u = \begin{pmatrix} -0.2115 \\ 0.5970 \end{pmatrix} \text{ and } c = 0.8030.$$

BACKPROP EXAMPLE: WEIGHT UPDATES

- Plugging all values into our model yields

$$f(\mathbf{x} \mid \boldsymbol{\theta}) = 0.7615$$

  and a squared error of

$$L(y, f(\mathbf{x})) = \frac{1}{2}(1 - 0.7615)^2 = 0.0284.$$

- The initial weights predicted $f(\mathbf{x}) = 0.7525$ and a slightly higher error value of $L(y, f(\mathbf{x})) = 0.0306$.

- Keep in mind that this is the result of only one training iteration. When applying a neural network, one usually conducts thousands of those.

BACKPROPAGATION EXAMPLE: SUMMARY

- Our goal was to minimize the true/expected risk

$$\mathcal{R}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L\left(y, f(\mathbf{x} \mid \boldsymbol{\theta})\right)\right]$$

  with respect to the true underlying distribution $\mathbb{P}_{xy}$.
- Because we do not know $\mathbb{P}_{xy}$, we decided to minimize the empirical risk

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

  w.r.t. the training set and hope for the best.
- However, even this is not possible because there is no way to analytically find a global minimum for deep neural networks.
- Therefore, we decided to use gradient descent to iteratively find a local minimum of the (typically) non-convex loss function.

BACKPROPAGATION EXAMPLE: SUMMARY

- To perform gradient descent, we want to compute the gradient of the loss function $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ with respect to **all** the weights and biases in the network.
- Therefore, the number of components in the gradient is the total number of weights and biases in the network.
- To compute each component in the gradient, we apply the chain rule to the relevant portion of the computational graph.
- In software implementations, a vectorized version of the chain rule is used to compute the derivatives w.r.t. multiple weights simultaneously.
- Loosely speaking, each term in the gradient represents the extent to which the corresponding weight is responsible for the loss. In other words, it is a way to assign "blame" to each weight in the network.

BACKPROPAGATION EXAMPLE: SUMMARY

- Gradient descent can be implemented efficiently:
  - We can store and reuse results of the forward pass
  - We can store and intermediate results due to the chain rule during the backward pass
  - We know how derivates of activation functions and affine functions look like

  For example, to compute the derivative of the sigmoid activation at $z_{1,out}$, we "stored" the derivative of the sigmoid function $\frac{\partial z_{1,out}}{\partial z_{1,in}} = \sigma(z_{1,in})(1 - \sigma(z_{1,in}))$ and plugged in $\sigma(z_{1,in}) = 0.3705$, which was known from the forward pass.

BACKPROPAGATION EXAMPLE: SUMMARY

- In our example, we updated w.r.t. a single training example but typically, we feed subsets of the training set (minibatch GD).
- The term “backpropagation” refers to the fact that the computation of the gradient using the chain rule is performed backwards (that is, starting at the output layer and finishing at the first hidden layer). A **forward** computation using the chain rule also results in the same gradient but can be computationally expensive (see http://colah.github.io/posts/2015-08-Backprop/).

BACKPROPAGATION EXAMPLE: SUMMARY

- After computing the gradient (using backpropagation), we subtract the gradient (scaled by the learning-rate α) from the current set of weights (and biases) which results in a new set of weights (and biases).
- The empirical loss for this new set of weights is now lower ("walking down the hill").
- Next, we once again compute the forward pass for this new set of weights and store the activations.
- Then, we compute the gradient of the empirical loss using backpropagation and take another step down the hill.
- Rinse and repeat (until the loss stops decreasing substantially). However, training until convergence often results in overfitting (use early stopping).