Data Science & Artificial Intelligence

Summer Term 2022

IFM Institut für Fahrerassistenz und vernetzte Mobilität

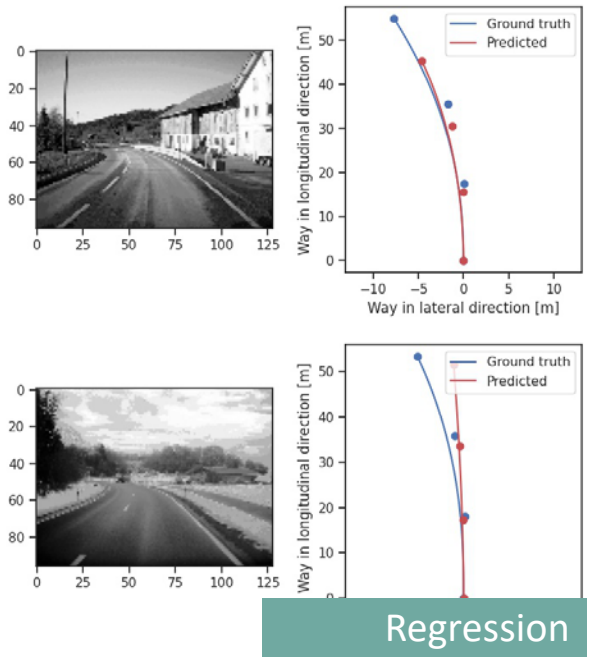
L08 Introduction Backpropagation

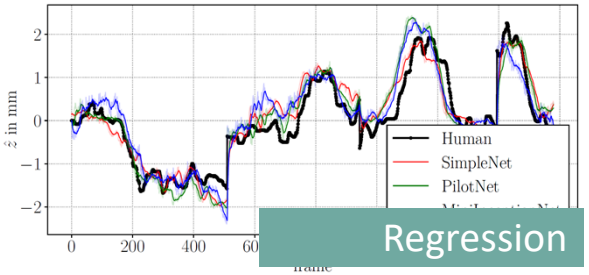
J. Haselberger, D. Schneider, B. Stuhr

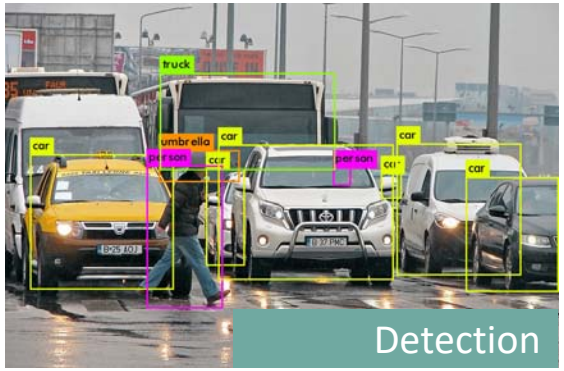
L08.1 Loss Functions - Motivation

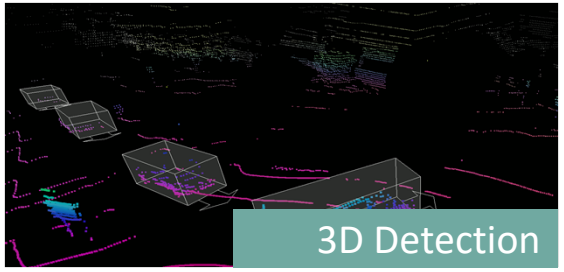
Why do we need a loss function?

- Independent of the task (regression, segmentation, detection …), we need to measure how close our predictions are to the ground truth label









L08.1 Loss Functions - Regression

Overview Regression metrics

- To calculate the performance, the given value y_i (ground truth, label), is compared with the prediction of the algorithm $\hat{y}_i$:

Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2$$

MSE: one of the most commonly used metrics

Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=0}^{n-1}|y_i - \hat{y}_i|$$

MAE: not very sensitive to outliers but useful if one bad prediction would ruin the predictive capability of the entire model

L08.1 Loss Functions - Regression

Overview Regression metrics

Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

RMSE: Extension of the mean squared error. The units of the RMSE are the same as the original units of the target value.

R^2 Score (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

R^2: also called Coefficient of determination. Describes the percentage of variation explained by the relationship between y_i and $\hat{y}_i$.

L08.1 Loss Functions - Classification

Overview Classification metrics

- One of the most important metrics of classification are Accuracy, Precision and Recall. To calculate these values, the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) are used.

Accuracy

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Accuracy: Of the total number of predictions made, how many did we predict correctly?

Precision

$$Precision = \frac{TP}{TP+FP}$$

Precision: What proportion of positive identifications were actually correct?

Recall

$$Recall = \frac{TP}{TP+FN}$$

Recall: What is the proportion of actual positive cases that were correctly identified?

L08.1 Loss Functions - Classification

Use Case Example

Use Case Example: Prediction of Cut-ins

- On a highway scenario, the model predicts whether a car is cutting into our lane or not.

type	number
True Cut-In (TP)	1
True no Cut-In (TN)	90
False Cut-In (FP)	1
False no Cut-In (FN)	8

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} = \frac{1+90}{1+90+1+8} = 91\%$$

➔ The total accuracy of the Cut-In detector is 91%

$$Precision = \frac{TP}{TP+FP} = \frac{1}{1+1} = 50\%$$

➔ the prediction that a cut-in is present is correct to 50%

$$Recall = \frac{TP}{TP+FN} = \frac{1}{1+8} = 11\%$$

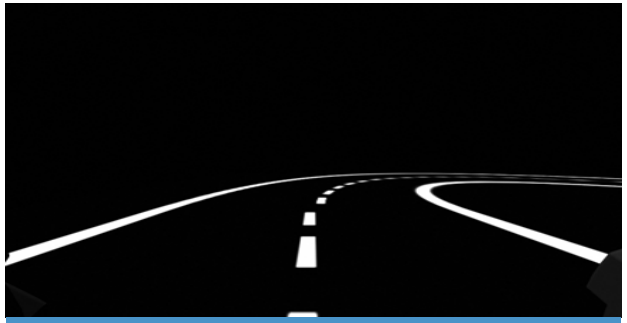
➔ The model correctly recognizes 11% of all cut-ins

? Why is the spread between Accuracy and Recall so wide?

L08.1 Loss Functions - Segmentation

Overview Segmentation metrics

- The loss function for segmentation models also depends on the nature of the task



binary

- **only one class** which pixels are marked with 1
- remaining pixels are background and are marked with 0

multiclass

- C = **1.. N classes** that have unique identification values
- **classes are mutually exclusive**
- all pixels are labeled

multilabel

- C = **1..N classes** which pixels are **labeled as 1**
- **classes are not mutually exclusive**
- **one channel per class**
- pixels in each channel that do **not belong to the class marked as 0**

L08.1 Loss Functions - Segmentation

Overview Segmentation metrics

Binary Cross-Entropy (BCE):

$$BCE = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

BCE: works best when the data is evenly distributed among the classes. Image masks with very strong class imbalance may not be evaluated appropriately by BCE (use weighted BCE). Loss is calculated pixel pairwise without knowing whether its neighboring pixels are boundaries or not.

DICE:

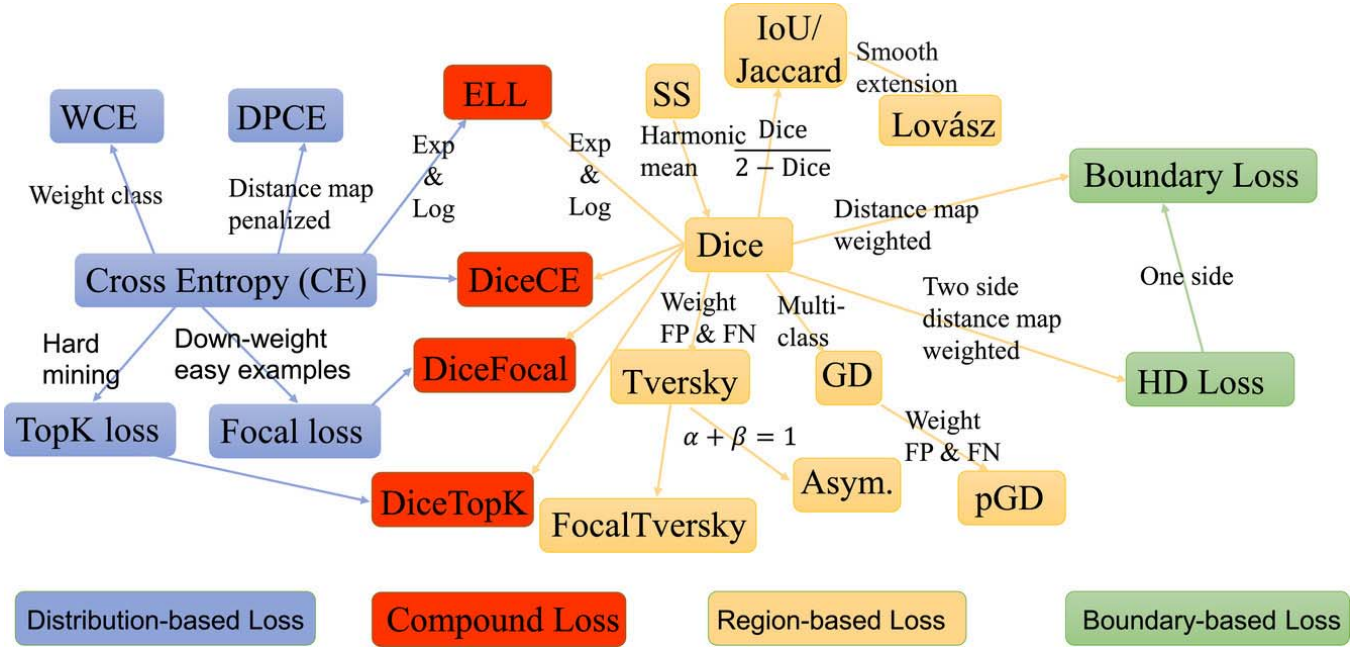
$$DICE = 1 - \frac{2y\hat{y}+1}{y+\hat{y}+1}$$

DICE: computes the similarity between images and is similar to the Intersection-over-Union heuristic. Dice loss considers the loss information both locally and globally, which is critical for high accuracy on boundaries.

L08.1 Loss Functions - Segmentation

Overview Segmentation metrics

- From the basic forms, specialized variations have emerged for various fields of application:



Loss odyssey in medical image segmentation [Ma2021]

L08.2 Introduction Gradient Descent

Intuition

- the model output $\hat{Y}$ depends on the trained network parameters Θ
- with the help of the loss function L the current performance can be determined

Model Prediction ($\hat{Y}$):

$$\hat{Y} = f(\Theta, X)$$

Parameter dependent loss (L):

$$L(\Theta) = L(Y, \hat{Y}) = L(Y, f(\Theta, X))$$

- The parameters Θ must be adjusted so that the loss function L is minimized for all data points: $\min_{\Theta} L(X, Y, \Theta)$

Iterative solution necessary
even small neural networks often have several million free parameters

L08.2 Introduction Gradient Descent

Hands-On LXX.XX.py

Gradient Descent

- Gradient Descent uses the first derivative to iteratively converge to the minimum
- gradient based methods have emerged as the most widely applied optimization algorithms in the field of deep learning [Sutskever2013]
- using small steps (η), the local error minimum is approached
- network parameters are adjusted against the gradient $\nabla_\Theta L(\Theta)$

Gradient Descent:

$$\Theta \rightarrow \Theta - \eta \nabla_\Theta L(Y, f(\Theta, X))$$

- $\nabla_\Theta L(\Theta)$ is a vector containing all partial derivations of the error function according to each individual network parameter

L08.2 Introduction Gradient Descent

Hands-On LXX.XX.py

Batch - vs. Stochastic - vs. Minibatch Gradient Descent

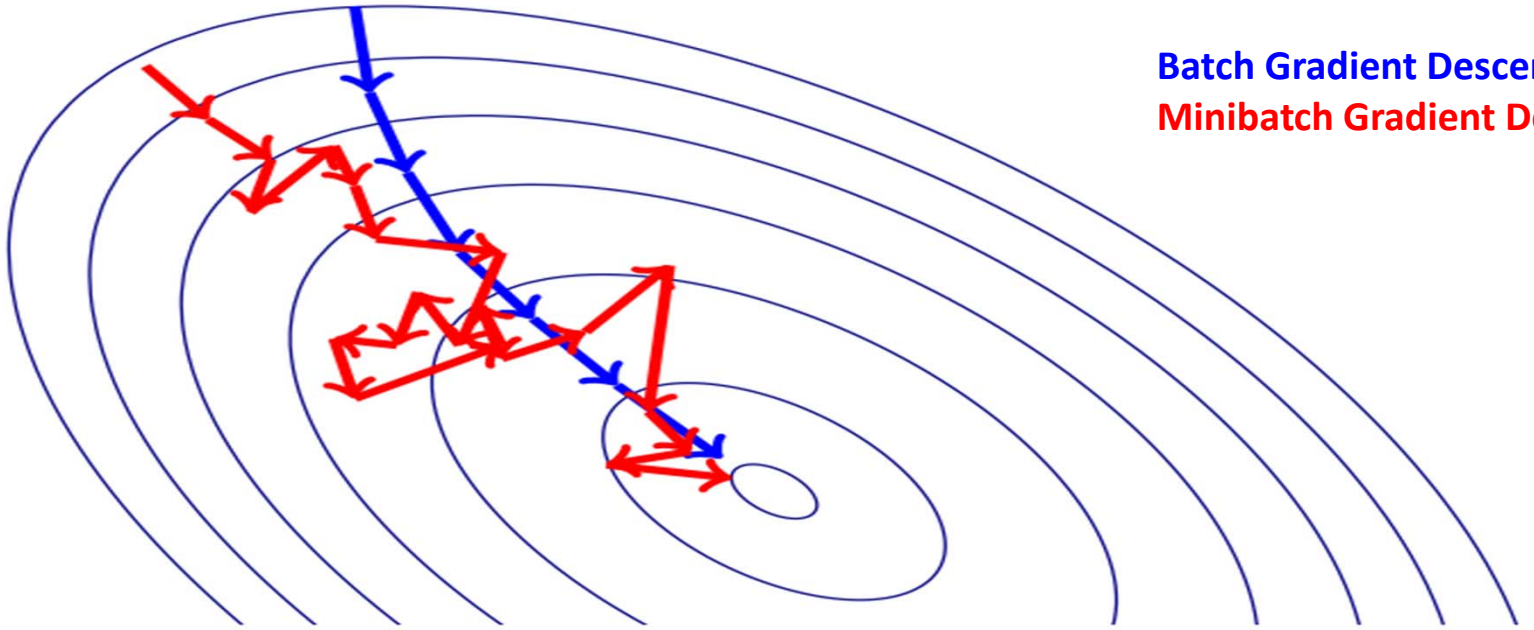
- There are different methods for calculating parameter updates depending on the number of data points used:

Batch Gradient Descent	Stochastic Gradient Descent	Minibatch Gradient Descent
• For a single update, **all available data** is used • stable • long training times • large datasets results in enormous memory requirements	• For a single update, **only one training sample** is used • faster calculation of an update step • large fluctuations during training	• **a small subset** of the training data is used for the update calculation • batch size defines the number of samples within each subset

Stochastic and Minibatch methods do not guarantee optimal convergence.
There is a risk that only one of many local minima will be found.

L08.2 Introduction Gradient Descent

Batch - vs. Stochastic - vs. Minibatch Gradient Descent



L08.3 Introduction Backpropagation

- Recap: $\nabla_\Theta L(\Theta)$ containins all partial derivations of the error function according to each individual network parameter
- even small networks have several thousand parameters: **how can the gradient be calculated for all these parameters?**

Building Block 1: Chain Rule

- the derivative of the composite function $y(x)$ is calculated from the product of the inner and outer derivative

Chain Rule:

$$y(x) = F\left(u(x)\right)$$

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx} = F'(u) * u'(x)$$

Building Block 2: Backpropagation

- also known as reverse mode automatic differentiation
- After each feed-forward pass, Backpropagation performs a backward pass using the Chain Rule

L08.3 Introduction Backpropagation

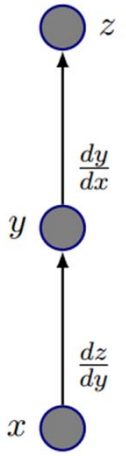
Chain Rule

- the derivative of the composite function $y(x)$ is calculated from the product of the inner and outer derivative

Chain Rule:

$$y(x) = F\left(u(x)\right)$$

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx} = F'(u) * u'(x)$$



Chain rule: $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$

L08.3 Introduction Backpropagation

Chain Rule

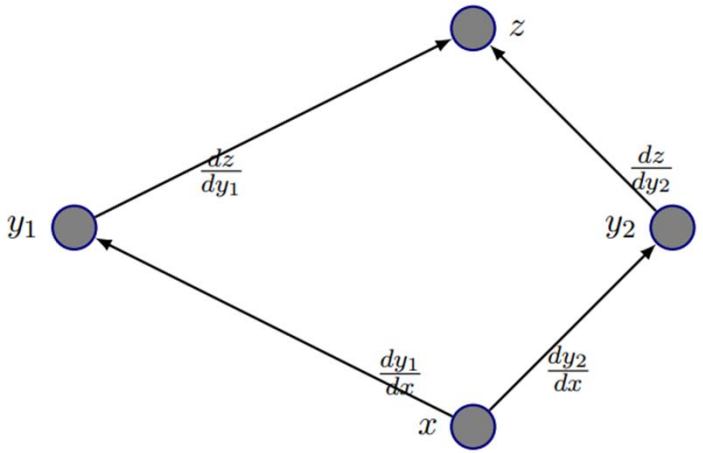
- the derivative of the composite function $y(x)$ is calculated from the product of the inner and outer derivative

Chain Rule:

$$y(x) = F\left(u(x)\right)$$

$$\frac{dy}{dx} = \frac{dy}{du} * \frac{du}{dx} = F'(u) * u'(x)$$

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

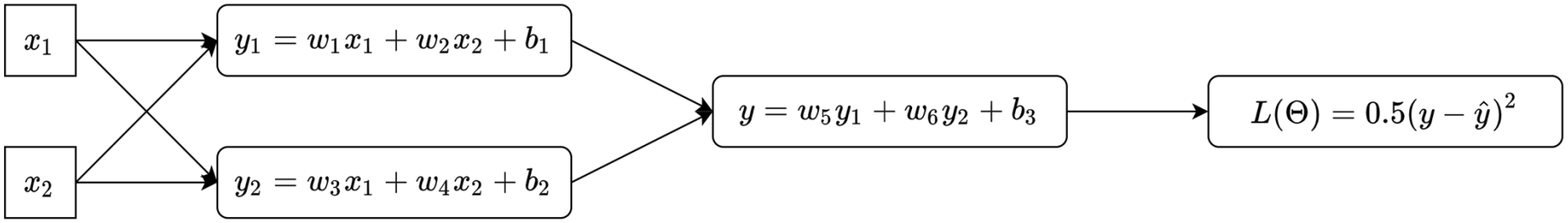


Multiple Paths: $\frac{dz}{dx} = \frac{dz}{dy_1}\frac{dy_1}{dx} + \frac{dz}{dy_2}\frac{dy_2}{dx}$

L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

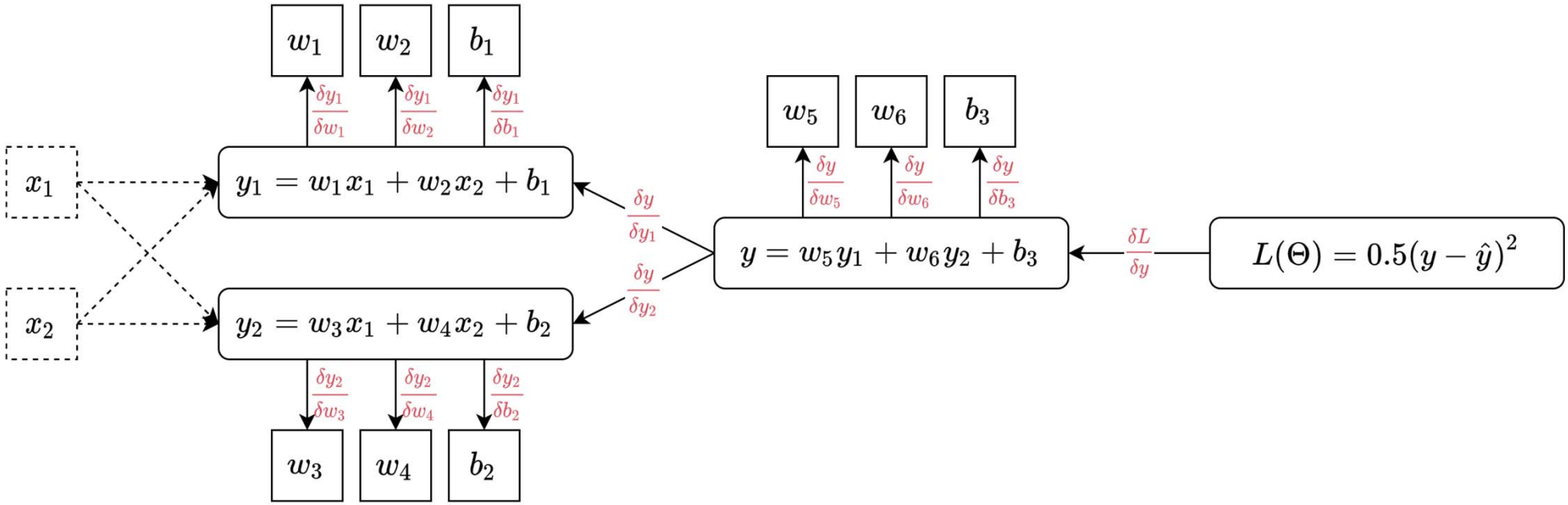
- we are interested in the parameter changes e.g. Δw_1 to reduce the loss $L(\Theta)$



L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

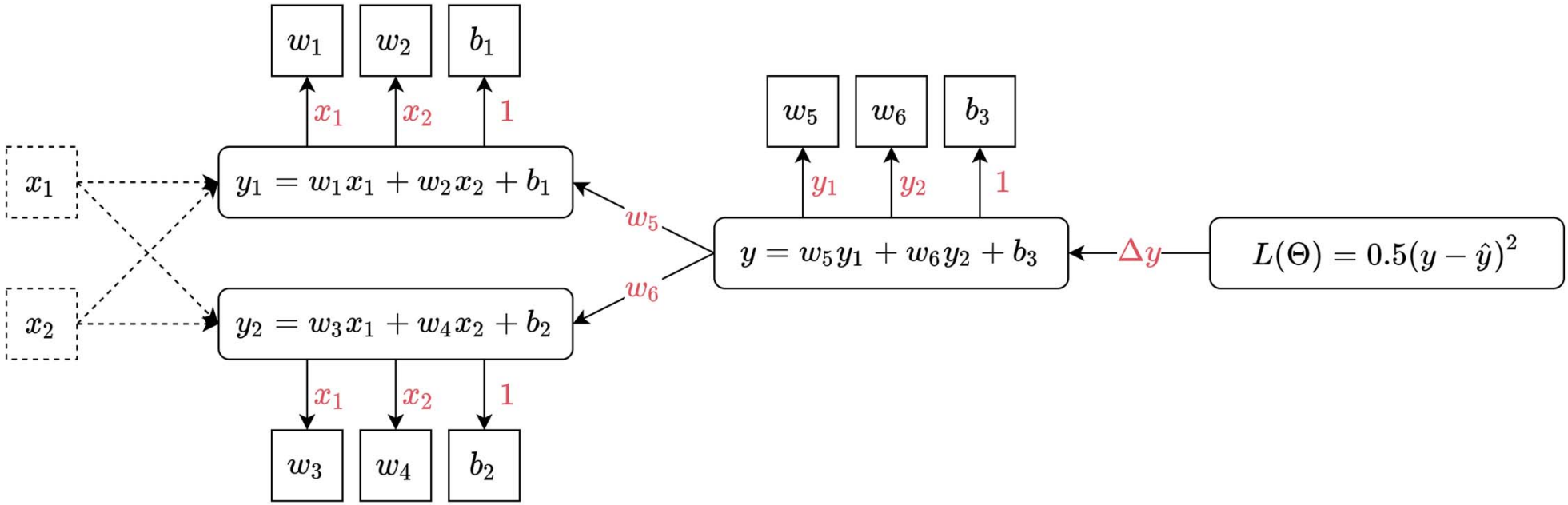
- First step: write down the local gradient, e.g. $\frac{\partial y_1}{\partial w_1}$



L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

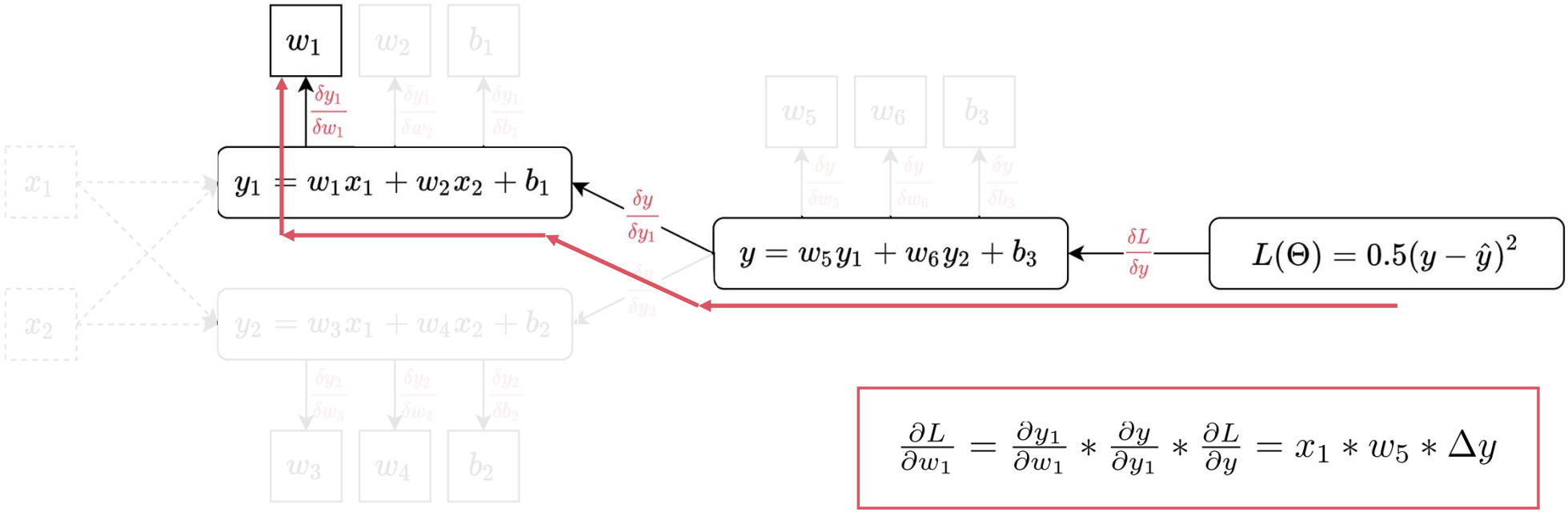
- Second step: calculate the local gradient, e.g. $\frac{\partial y_1}{\partial w_1} = x_1$



L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

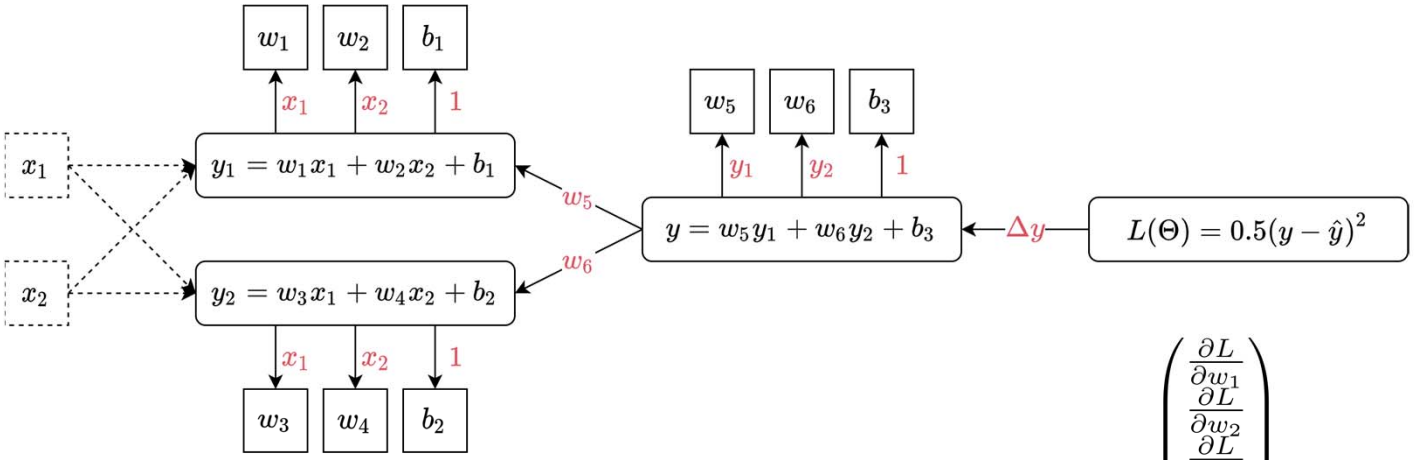
- Third step: back propagate through the network using the chain rule



L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

- Third step: back propagate through the network **for every parameter**



$$\nabla_{\Theta_t} L(\Theta_t) = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial w_3} \\ \frac{\partial L}{\partial w_4} \\ \frac{\partial L}{\partial w_5} \\ \frac{\partial L}{\partial w_6} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \\ \frac{\partial L}{\partial b_3} \end{pmatrix} = \Delta y * \begin{pmatrix} w_5 x_1 \\ w_5 x_2 \\ w_6 x_1 \\ w_6 x_2 \\ y_1 \\ y_2 \\ w_5 \\ w_6 \\ 1 \end{pmatrix} = \Delta y * \begin{pmatrix} w_5 x_1 \\ w_5 x_2 \\ w_6 x_1 \\ w_6 x_2 \\ w_1 x_1 + w_2 x_2 + b_1 \\ w_3 x_1 + w_4 x_2 + b_2 \\ w_5 \\ w_6 \\ 1 \end{pmatrix}$$

L08.3 Introduction Backpropagation

Use Case Example: MLP with two inputs, MSE loss

- Fourth step: perfrom the network update

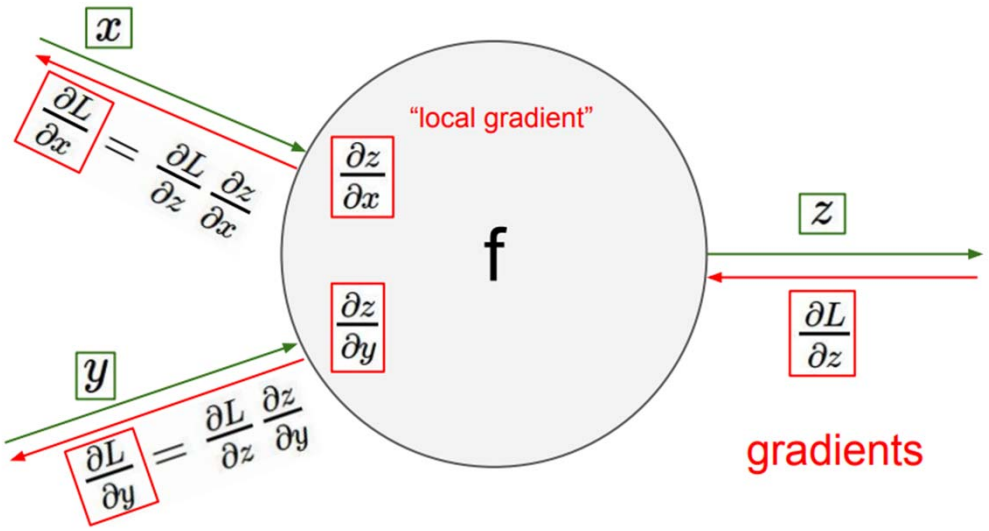
$$\Theta \rightarrow \Theta - \eta \nabla_{\Theta} L(Y, f(\Theta, X))$$

$$\nabla_{\Theta_t} L(\Theta_t) = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial w_3} \\ \frac{\partial L}{\partial w_4} \\ \frac{\partial L}{\partial w_5} \\ \frac{\partial L}{\partial w_6} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial b_2} \\ \frac{\partial L}{\partial b_3} \end{pmatrix} = \Delta y * \begin{pmatrix} w_5 x_1 \\ w_5 x_2 \\ w_6 x_1 \\ w_6 x_2 \\ y_1 \\ y_2 \\ w_5 \\ w_6 \\ 1 \end{pmatrix} = \Delta y * \begin{pmatrix} w_5 x_1 \\ w_5 x_2 \\ w_6 x_1 \\ w_6 x_2 \\ w_1 x_1 + w_2 x_2 + b_1 \\ w_3 x_1 + w_4 x_2 + b_2 \\ w_5 \\ w_6 \\ 1 \end{pmatrix}$$

L08.3 Introduction Backpropagation

Automatic Differentiation

- Computation of the gradient can be automatically inferred
- Every node needs to know:
  - how to compute its output
  - how to **compute its gradient with respect to its inputs given the gradient w.r.t its outputs**
- If we recompute the derivates each time, computation time can blow up!
- Need to **book-keep derivatives as we go down the network and reuse them**

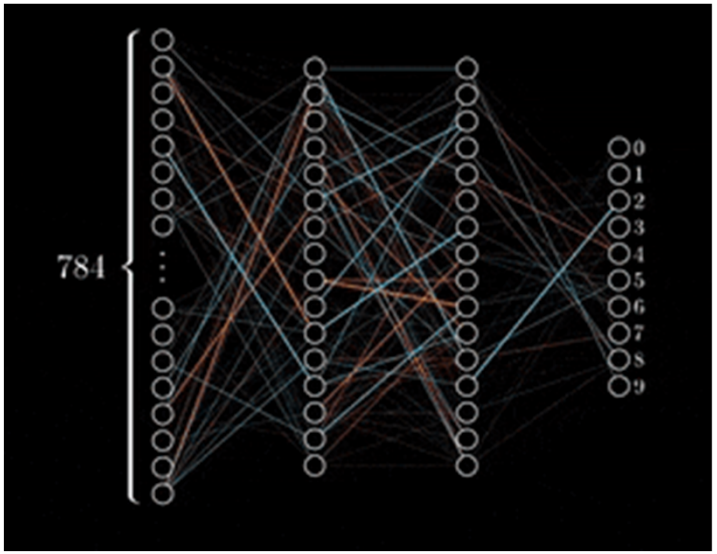


L08.3 Summary up this point

Neural Nets can be very large: impractical to write down all the gradient formulas by hand

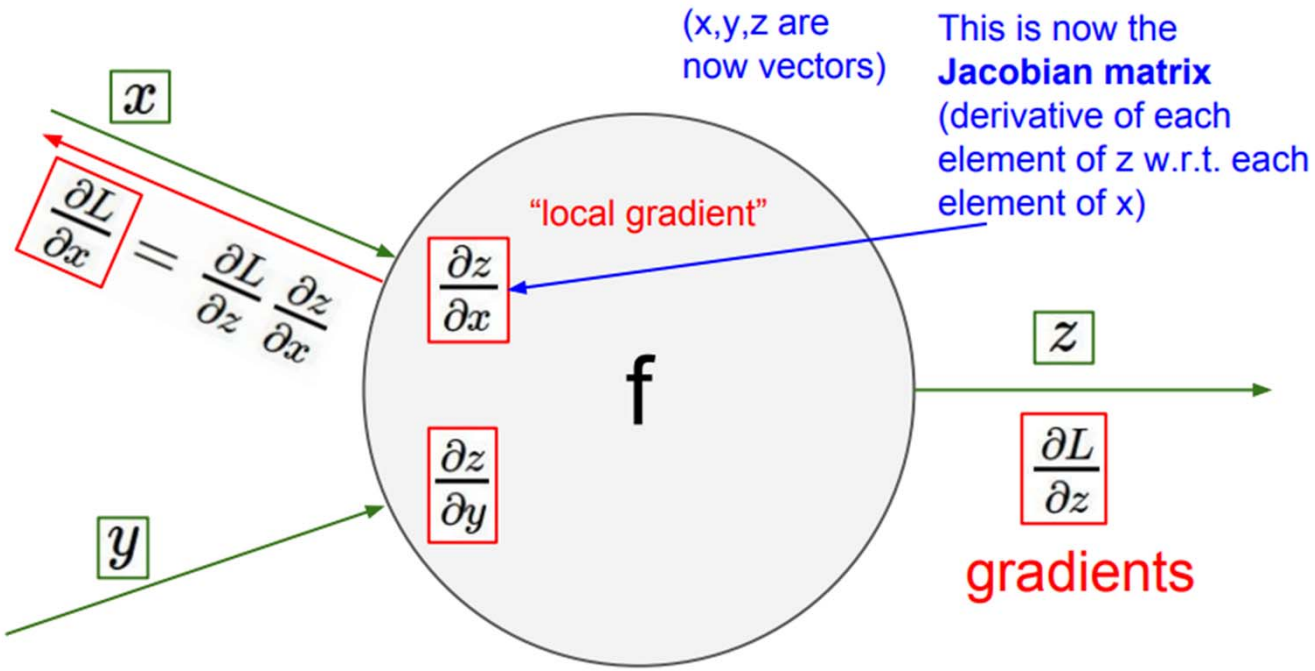
Backpropagation in the context of neural networks is about allocating credits that correspond to the weights:

- follow the path from the output to the edge we want to optimize
- find the derivations from the top to the edge by using the chain rule
- apply the gradient descent rule



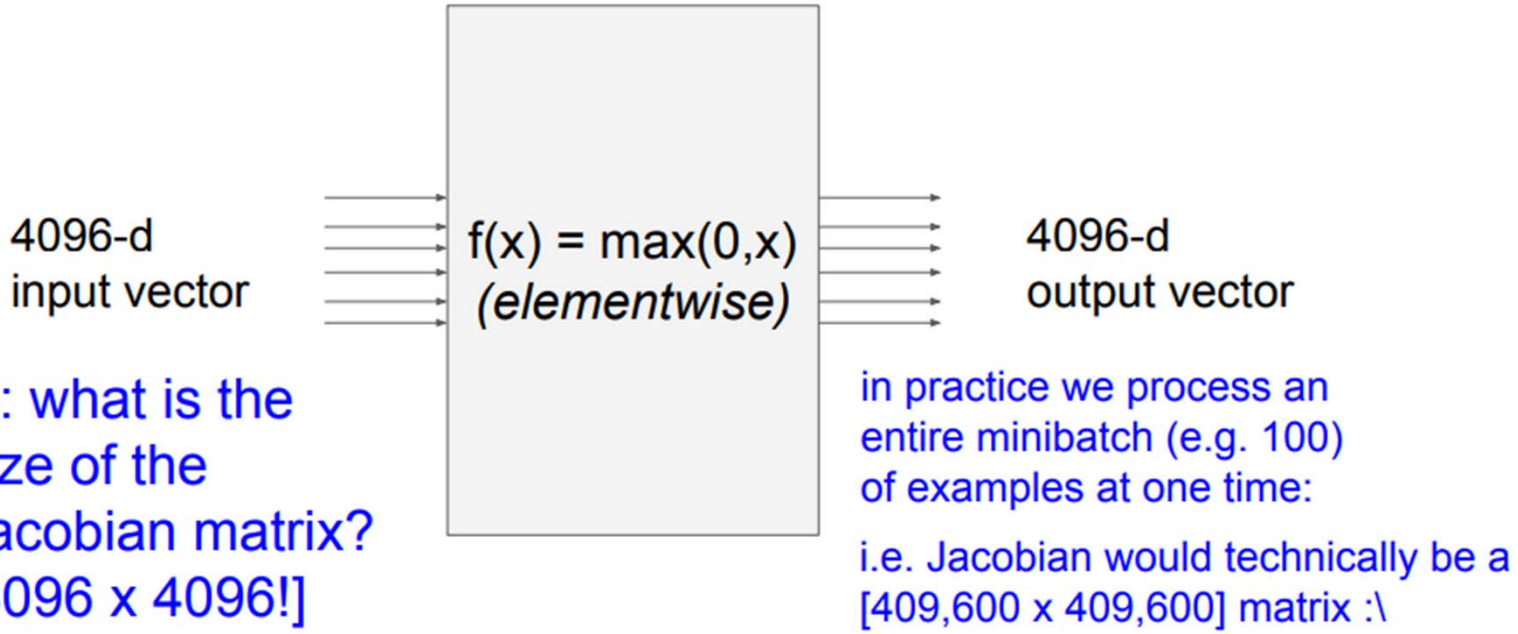
L08.3 Introduction Backpropagation

Vectorized operations (x and y are now vectors)



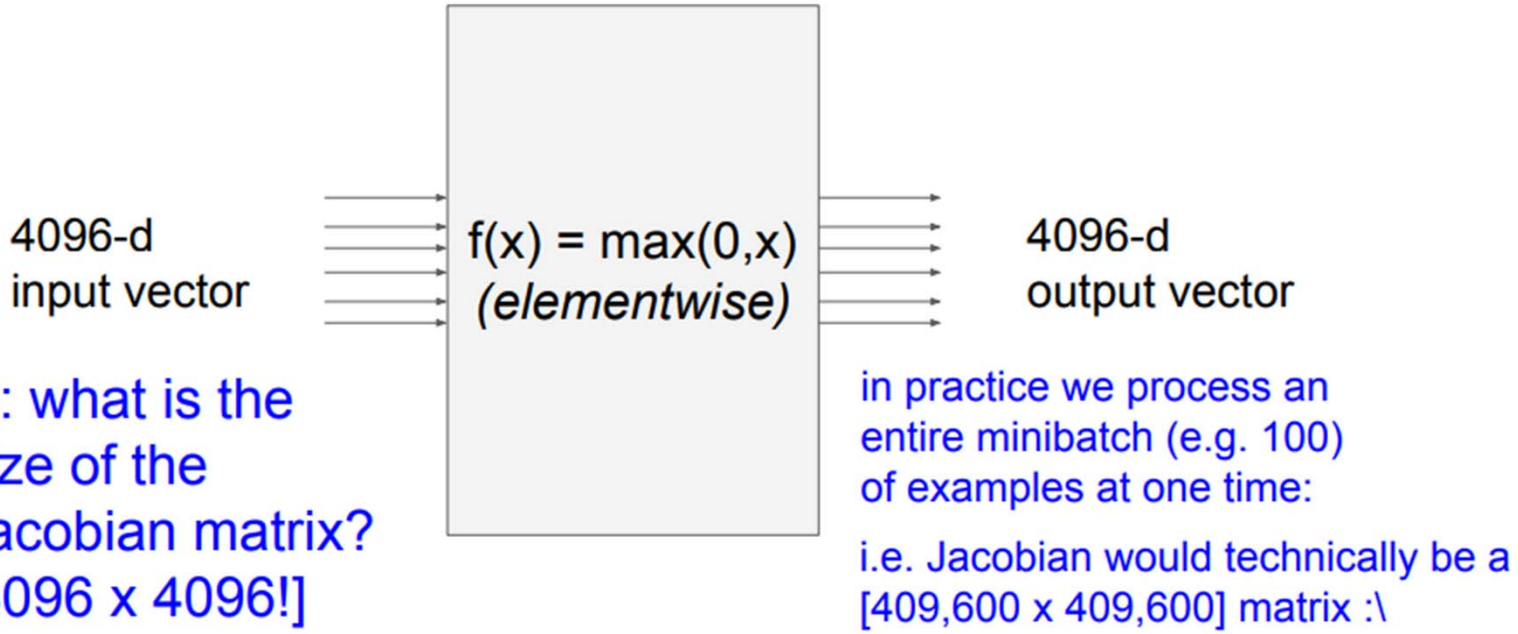
L08.3 Introduction Backpropagation

Vectorized operations



L08.3 Introduction Backpropagation

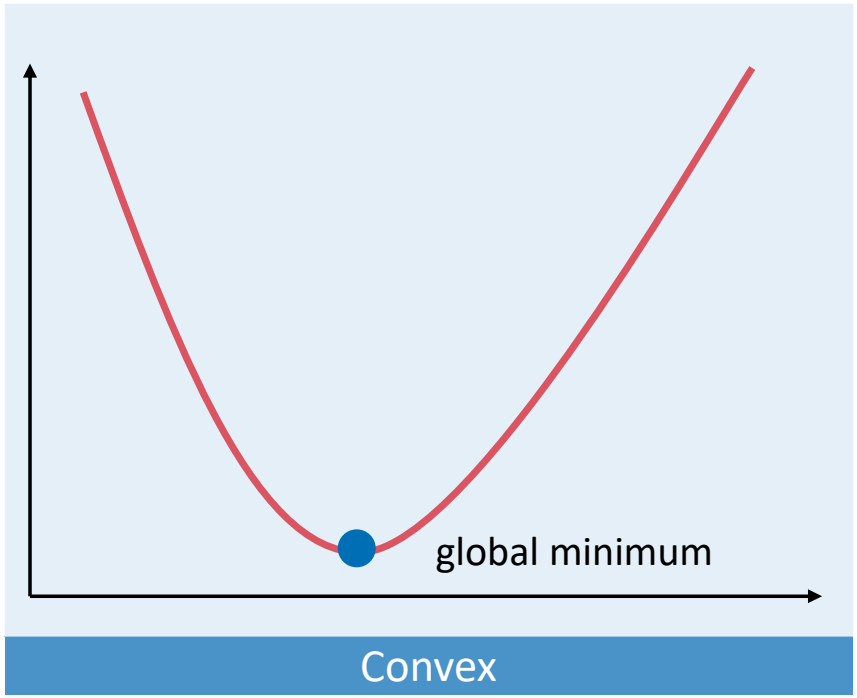
Vectorized operations

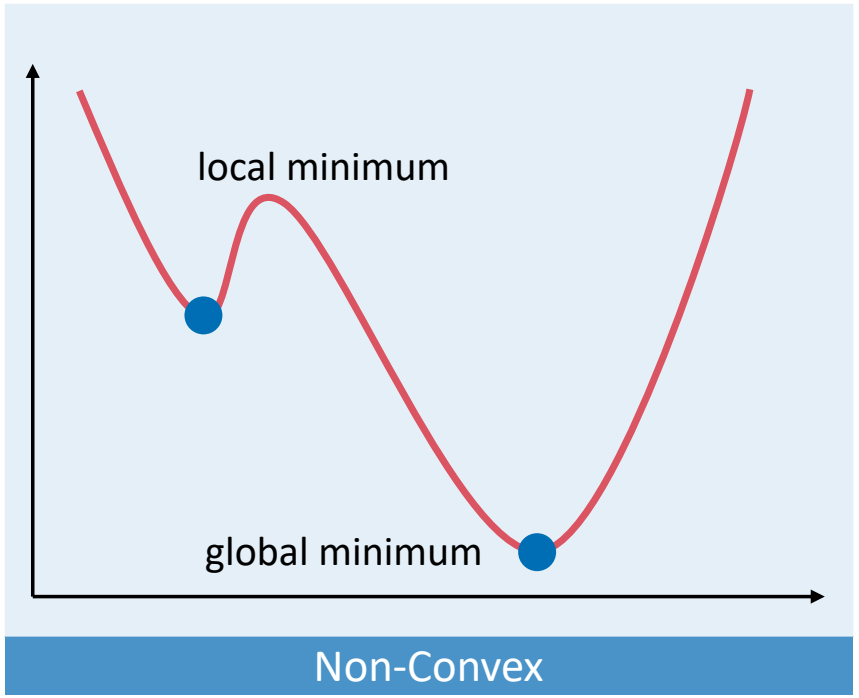


Practical usage:
We never form the full Jacobian matrix. Sparse structure ➔ handcode operations

L08.3 Introduction Backpropagation

Convex Vs. Non-Convex Error Surface





L08.4 Improved Descent Methods

Momentum

- Momentum procedure can be compared to a ball rolling down a hill
- As the ball rolls, it builds up momentum, thus increasing its speed
- helps to skip local minima

Gradient Descent with Momentum:

$$v_t = \underbrace{\eta \nabla_\Theta L(\Theta)}_{plain\ descent} + \underbrace{\gamma v_{k-1}}_{momentum}$$

$$\Theta \rightarrow \Theta - v_t$$

! If the momentum is too large, even the global minimum will be skipped because it does not decay fast enough.

L08.4 Improved Descent Methods

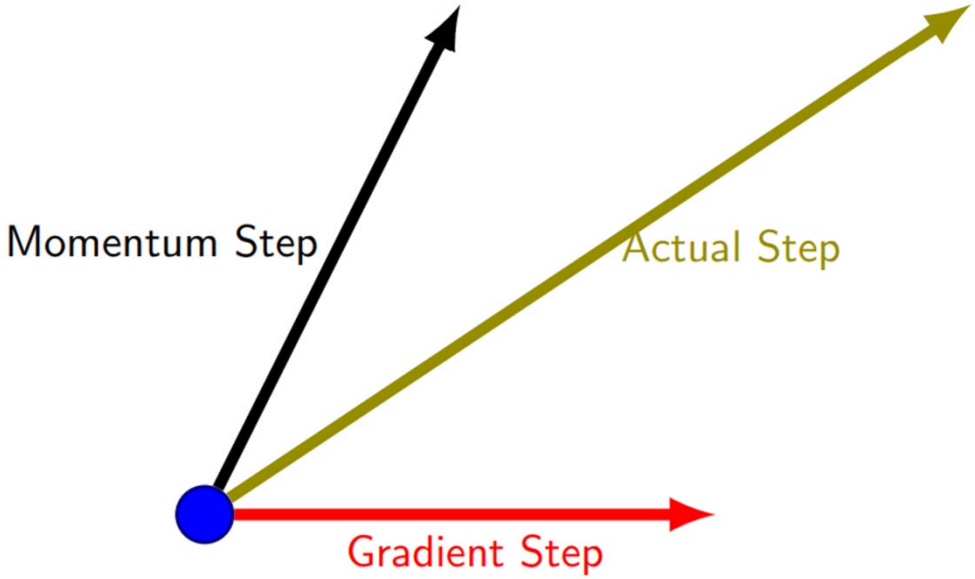
Momentum

Gradient Descent with Momentum:

$$v_t = \underbrace{\eta \nabla_\Theta L(\Theta)}_{plain\ descent} + \underbrace{\gamma v_{k-1}}_{momentum}$$

$$\Theta \rightarrow \Theta - v_t$$

Close analogy:
A ball (or panda) rolls down a hill, gathering momentum as it goes



L08.4 Improved Descent Methods

Momentum

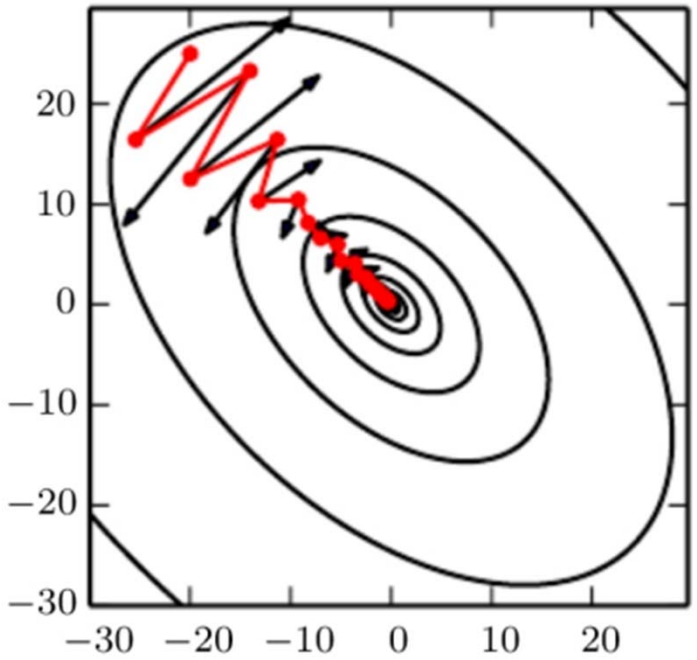


Illustration of how the momentum better navigates through such a error surface compared to plain gradient descent

L08.4 Improved Descent Methods

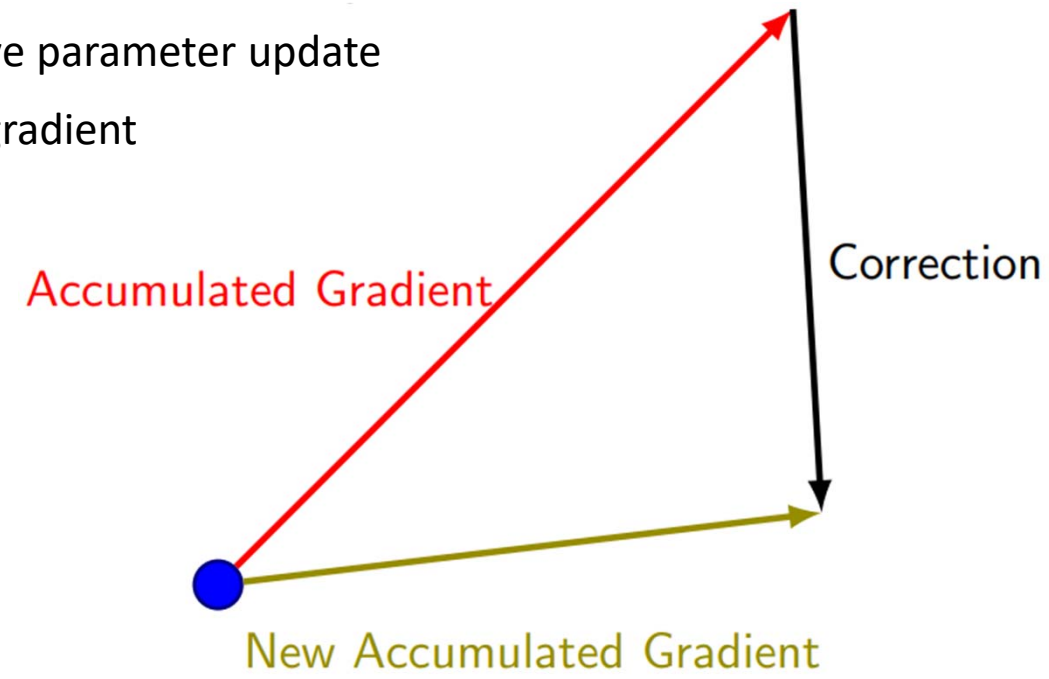
Nesterov Accelerated Gradient (NAG)

- Extension of the Momentum method with a predictive parameter update
- First take a step in the direction of the accumulated gradient
- Then calculate the gradient and make a correction

Nesterov (NAG):

$$v_t = \eta \nabla_\Theta L(\underbrace{\Theta - \gamma v_{k-1}}_{preview}) + \underbrace{\gamma v_{k-1}}_{momentum}$$

$$\Theta \rightarrow \Theta - v_t$$



! Strong dependence on selected learning rate!

L08.4 Improved Descent Methods

Adaptive Moment Estimation (ADAM)

- calculates specific learning rates for each individual parameter in addition to a momentum
- exponentially decaying average values of past gradients, as well as their squares are stored

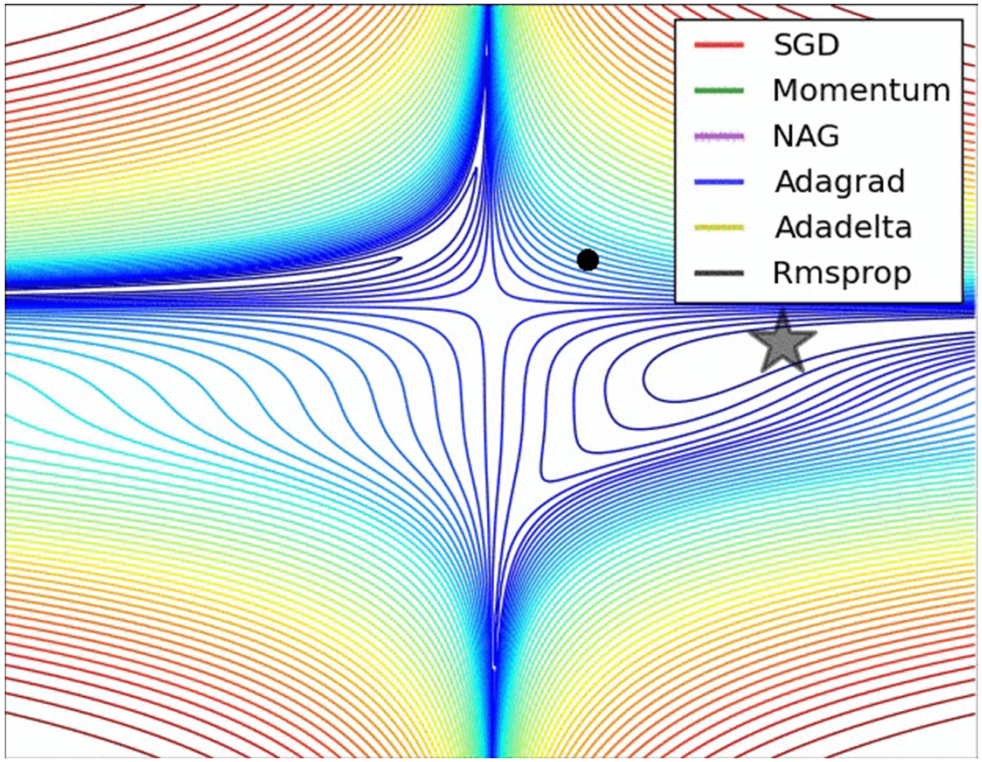
Adaptive Moment Estimation (ADAM):

$$m_t = \beta_1 m_{k-1} + (1-\beta_1)\nabla_{\Theta_k} L(\Theta_k) \qquad \hat{m}_k = \frac{m_k}{1-\beta_1^k}$$

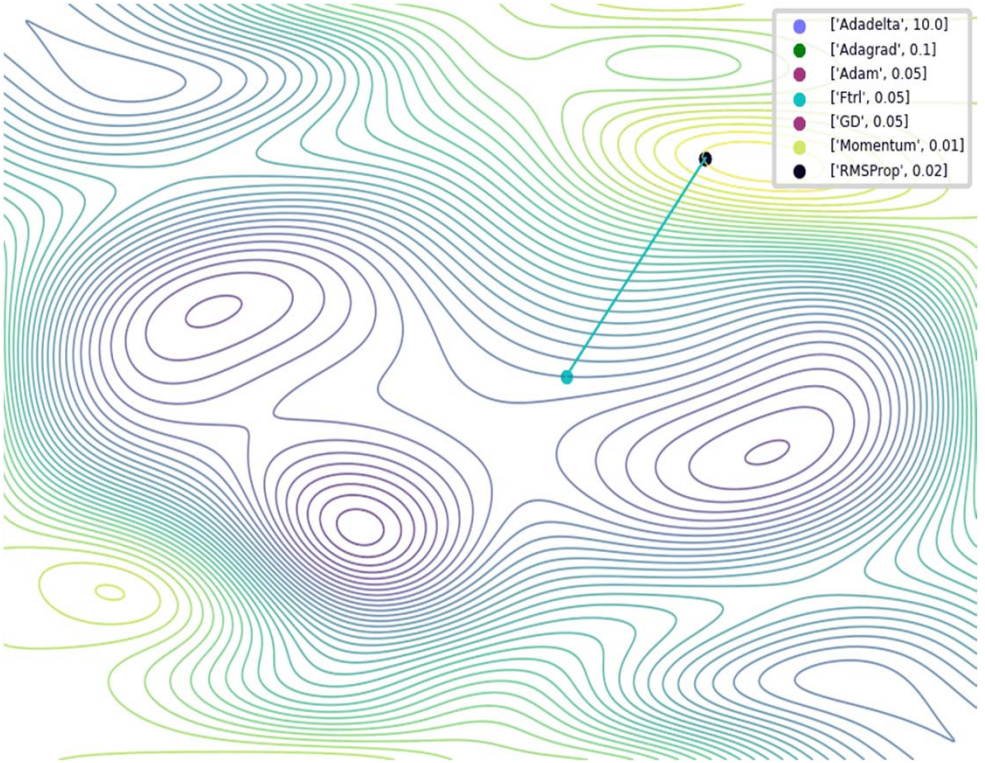
$$v_t = \beta_2 v_{k-1} + (1-\beta_2)\left(\nabla_{\Theta_k} L(\Theta_k)\right)^2 \qquad \hat{v}_k = \frac{v_k}{1-\beta_2^k}$$

$$\Theta \rightarrow \Theta - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon}\hat{m}_k$$

L08.4 Improved Descent Methods



[Ruder2016]



[Yun2018]

Dependency on loss landscape
There is no guarantee that we will find the global minimum!

L08.5 Bias-Variance Trade-Off

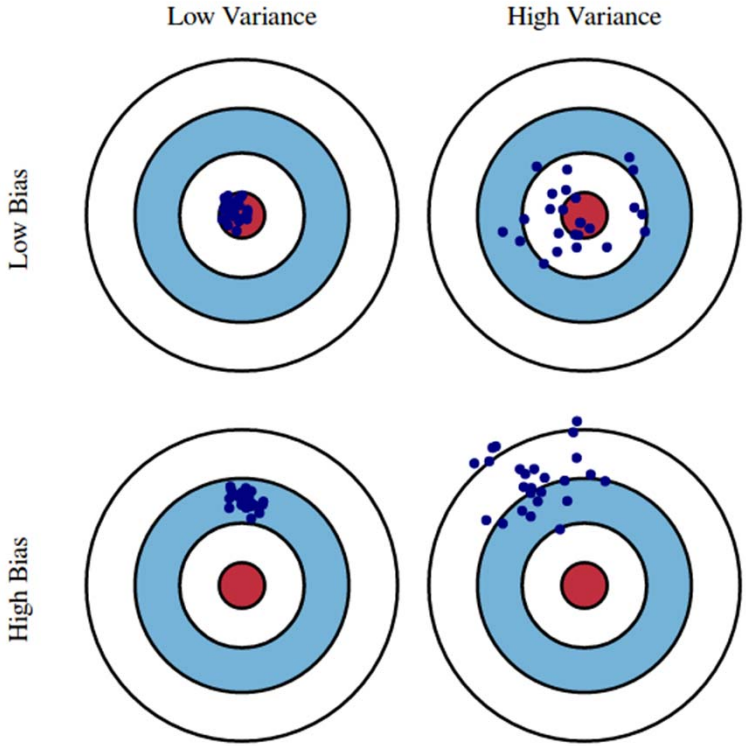
Bias

- difference between the average prediction and the correct value
- model with **high bias pays very little attention to the training data** and oversimplifies the model

Variance

- variability of prediction for a given data point. Value which tells the spread of the data
- model with **high variance pays a lot of attention to training data** and does not generalize on the data which it hasn't seen before [Singh2018]

Both low bias and low variance
large, high quality data sets required



Graphical illustration of bias and variance from [FortmannRoe2012]

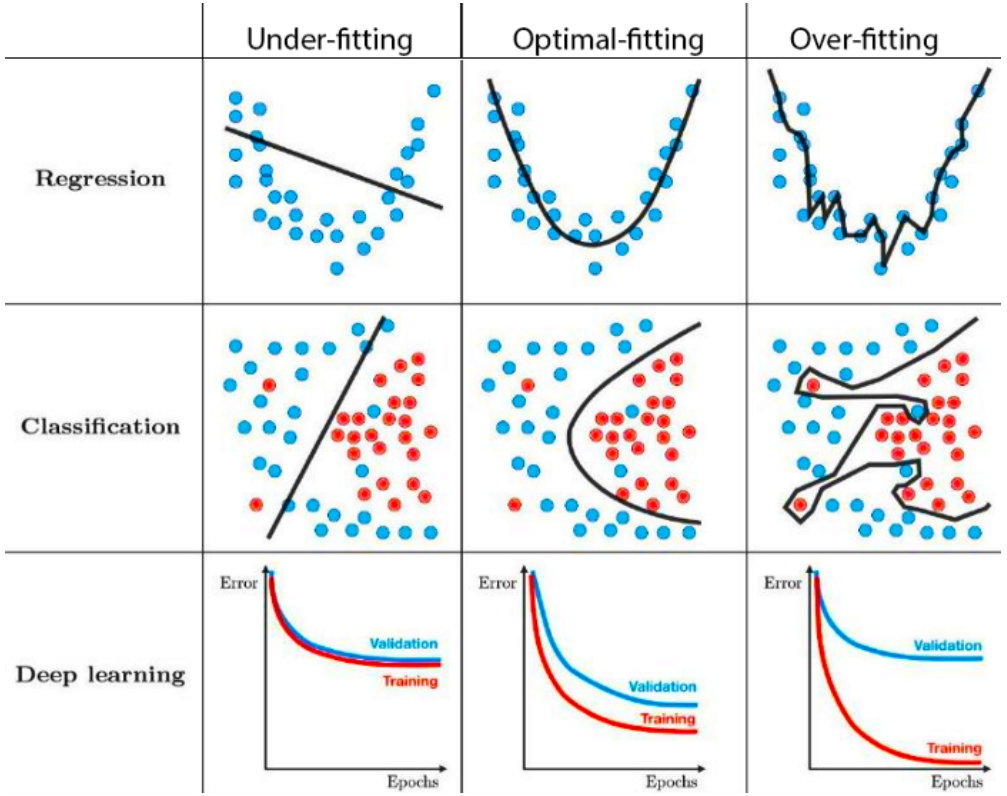
L08.5 Over- and Underfitting

Underfitting

- model is not complex enough for the underlying data
- **high bias**

Overfitting

- model is too complex and even captures the noise in the data
- **high variance**



L08.5 Prevent Overfitting

Regularization

- penalize high coefficients (weights) during optimization
- weighting (λ) of penalty term gives intuitive hyperparameter to control model complexity

Ridge Regression (L2):

$$\underbrace{\min_{\Theta} L(X, Y, \Theta)}_{default} + \underbrace{\lambda \Theta^2}_{L2}$$

L2: good method to prevent overfitting. Difficult to apply in high dimensional feature spaces.

Lasso Regression (L1):

$$\underbrace{\min_{\Theta} L(X, Y, \Theta)}_{default} + \underbrace{\lambda \sum_i |\theta_i|}_{L1}$$

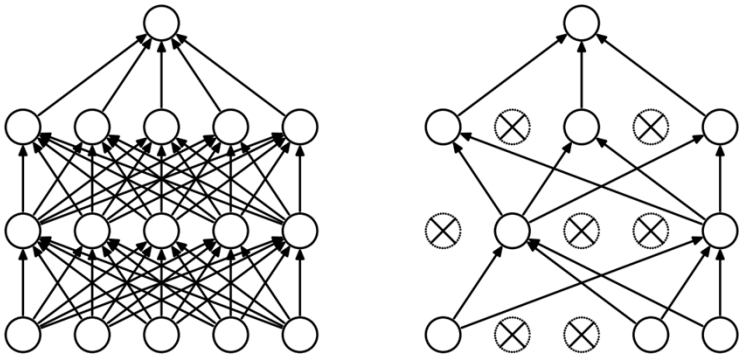
L1: tends to produce sparse solution (several coefficients go to zero). Can be applied for feature selection.

$$\vec{\Theta} = [\theta_1, 0, 0, \theta_4, \theta_5]$$

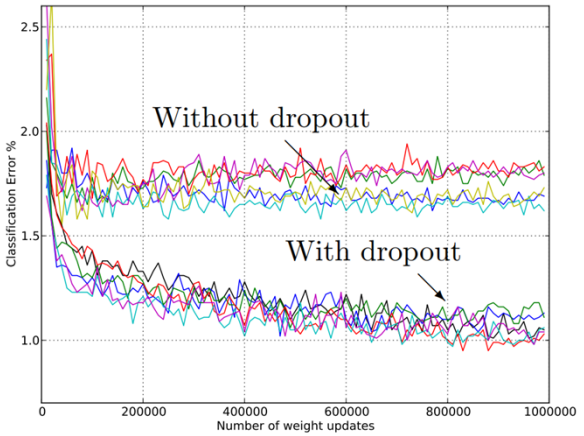
L08.5 Prevent Overfitting

Dropout

- Standard backpropagation learning builds brittle coadaptations that work for training data but cannot be generalized to unseen data
- Dropout randomly interrupts connections between individual neurons
- by making the presence of particular hidden entity unreliable, these co-adaptations are resolved



(a) Standard Neural Net (b) After applying dropout.



Basic Dropout principle and influence on training process [Srivastava2014]

L08.6 Initialization

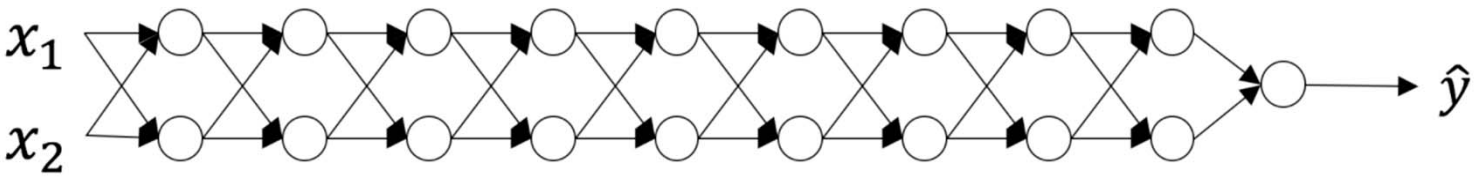
The importance of effective initialization

- For convex problems with good learning rate, convergence is guaranteed regardless of initialization

- For non-convex problems, the initialization step can be critical to the model's ultimate performance, and it requires the right method
- Neural Networks are not well understood to have principled, mathematically nice initialization strategies
- Most initialization strategies are based on intuitions and heuristics

L08.6 Exploding or Vanishing Gradients

- At each iteration of the optimization loop (forward, cost, backward, update), it can be observed that the backpropagated gradients are either amplified or minimized as one moves from the output layer to the input layer



Example of a neural network with 9 layers

A too-large initialization leads to exploding gradients

- leads the cost to oscillate around its minimum value

A too-small initialization leads to vanishing gradients

- The gradients of the cost with respect to the parameters are too small, leading to convergence of the cost before it has reached the minimum value

L08.6 Initialization

Rules of thumb to prevent exploding or vanishing gradients

- The mean of the activations should be zero
- The variance of the activations should stay the same across every layer

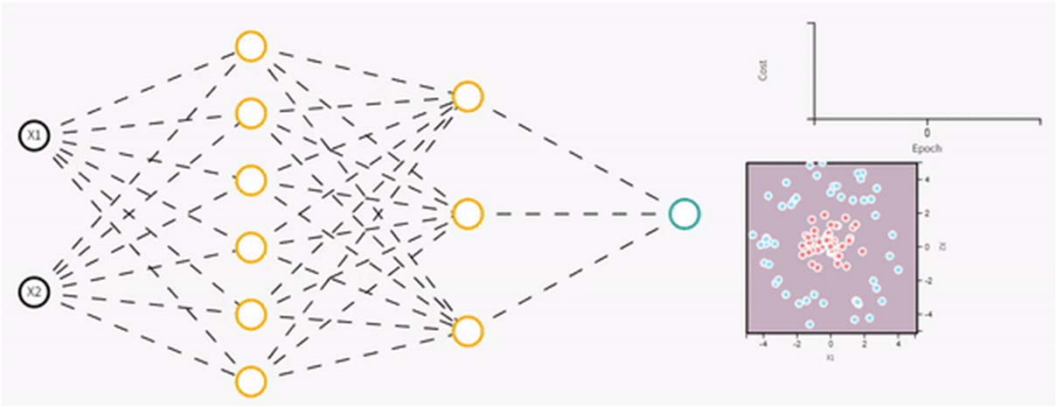
➔**The recommended initialization is Xavier initialization (or one of its derived methods)**

- **Notice:** Xavier initialization works with tanh activations. For ReLU a common initialization is the He initialization

L08.6 Initialization

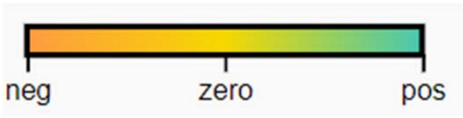
Does initialization matter?

- Example I: all weights are initialized with zero



- any constant initialization scheme will perform very poorly
- neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things

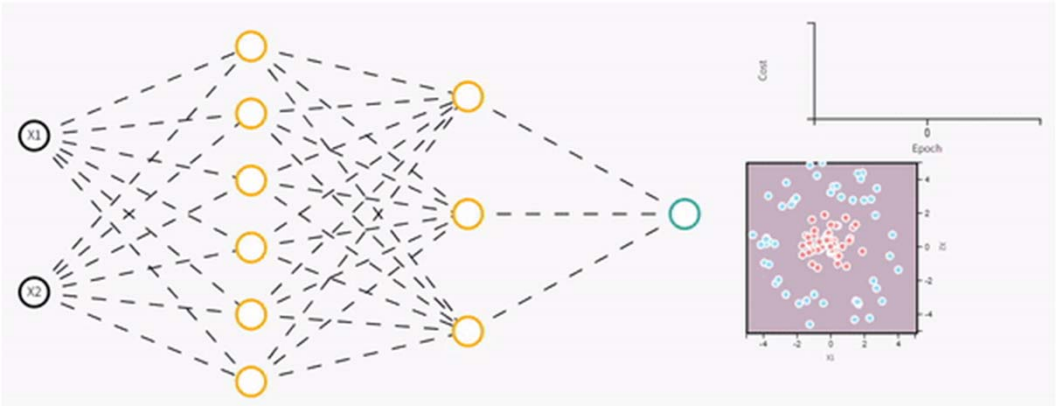
Gradients



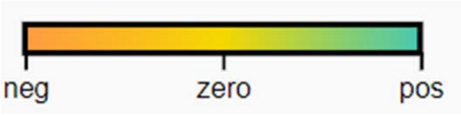
L08.6 Initialization

Does initialization matter?

- Example II: all weights are initialized too small



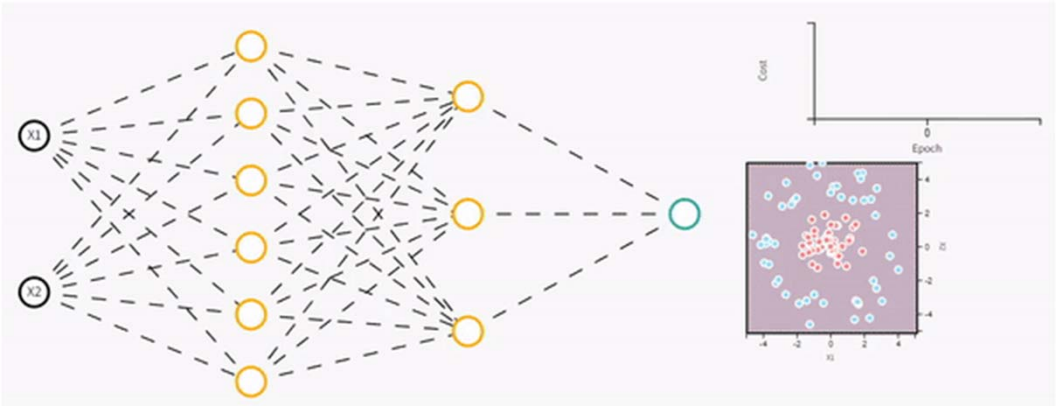
Gradients



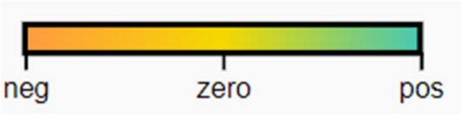
L08.6 Initialization

Does initialization matter?

- Example III: all weights are initialized just right



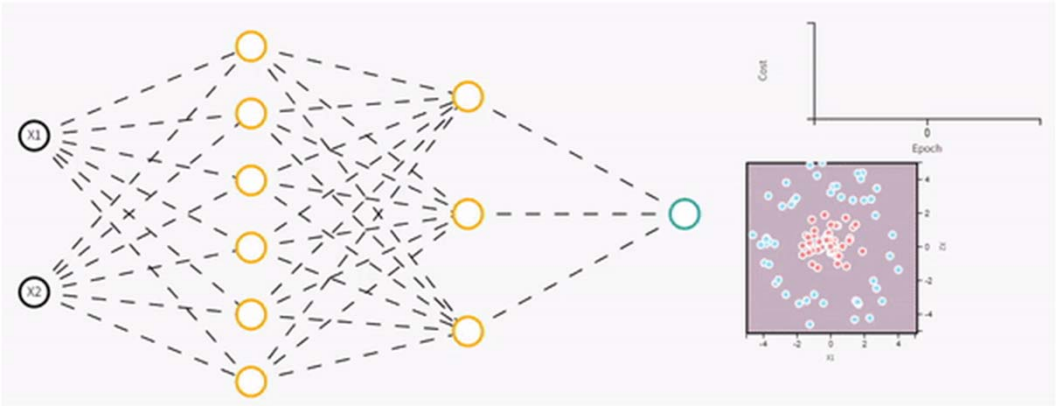
Gradients



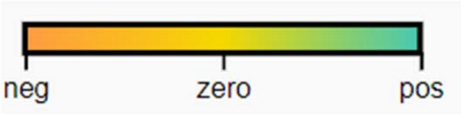
L08.6 Initialization

Does initialization matter?

- Example IV: all weights are initialized too high



Gradients



www.hs-kempten.de/ifm