Neural Networks 1/3

Lecture 10

Computer Vision for Geosciences

May 21, 2021

Universidad Nacional Autónoma de México

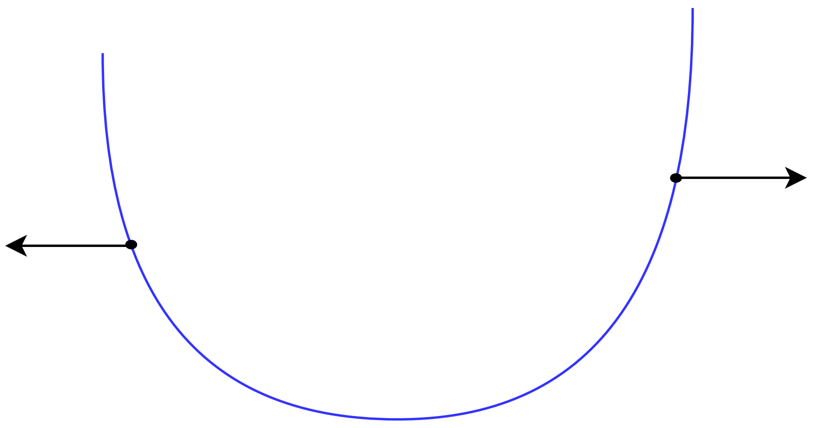
- The introduced formulation for Logistic Regression has no analytical solution.
- We can search for minima by walking on the error surface in the direction of steepest decent.

$$p(c|x) = y(x) = \sigma(Wx), \quad \sigma_i(x) = \frac{e^{x_i}}{\sum_{\forall j} e^{x_j}}$$

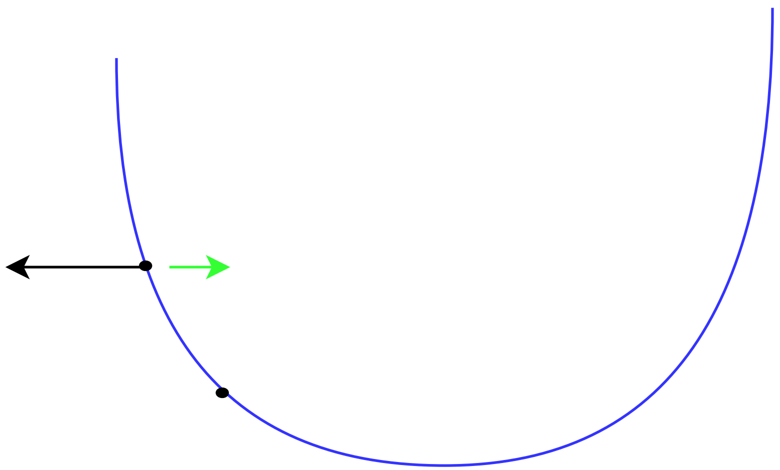
$$\nabla E(\theta) = \sum_i (y_i - t_i) x_i$$

$$\nabla E(\theta) = \sum_i (y_i - t_i) x_i$$

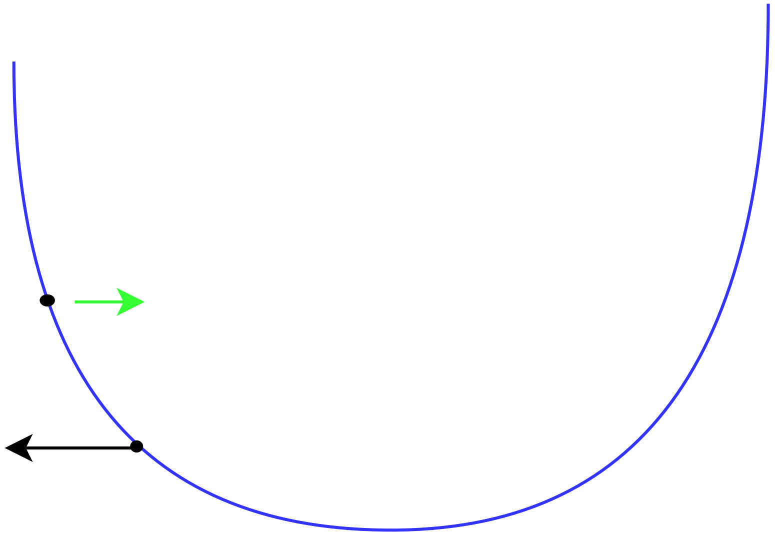
- We start at a random point and search for a minimum by walking on the error surface in the direction of steepest decent.



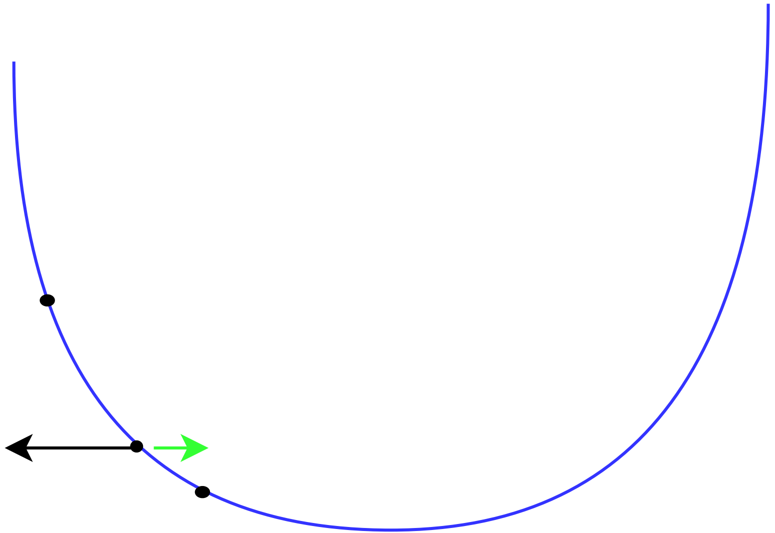
- For the error $E(\Theta)$ blue, the gradient points into the direction of steepest ascent.



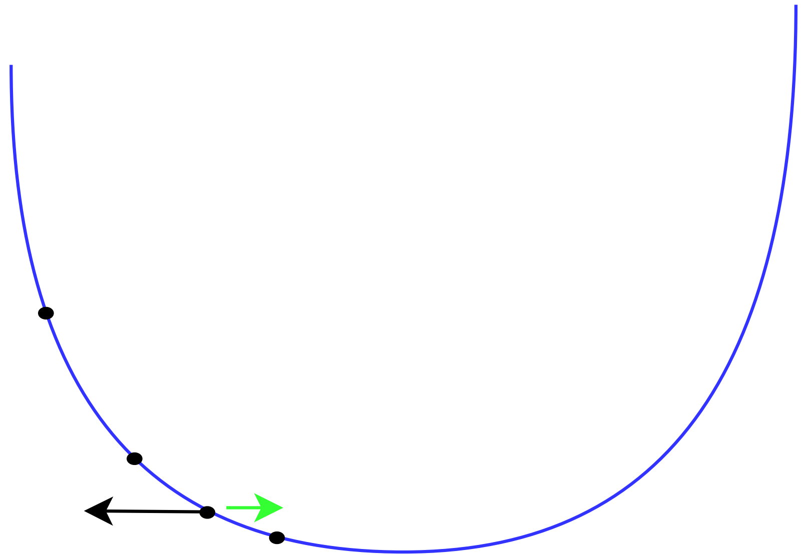
- Given a random initialization for θ we can evaluate the derivative and move into opposite direction.



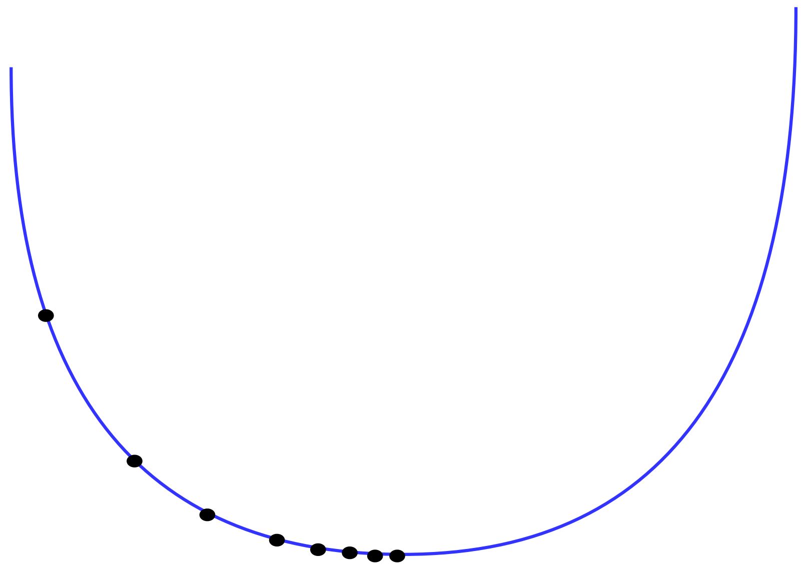
- We repeat the procedure at the new θ.



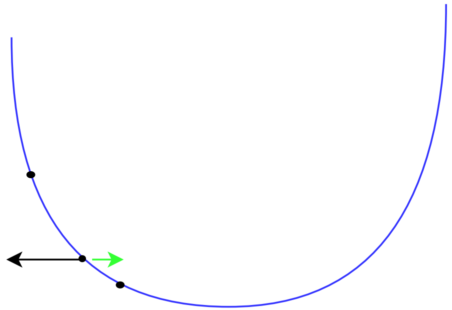
- We repeat the procedure at the new θ.



- We repeat the procedure at the new θ.



- And end up at a local minimum.



$$\theta_{i+1} = \theta_i - \eta \nabla E(\theta)$$

- We can write the update step formally including the learning rate (step size) η.
- Whereas ∇ is the gradient operator.

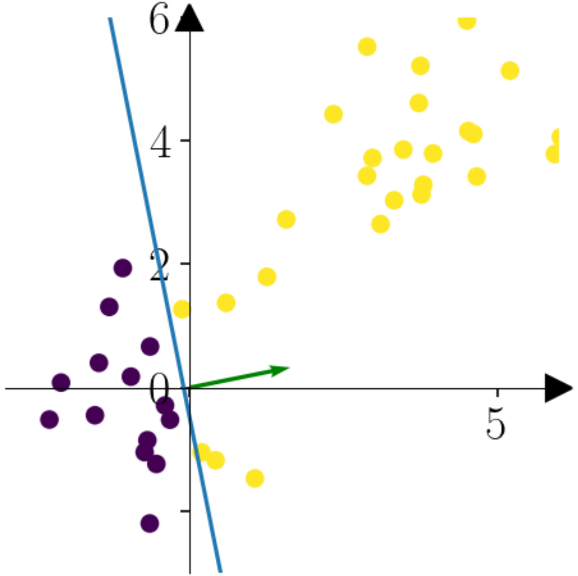
$$E(\theta) = \sum_i E_i(\theta)$$

$$\theta_{i+1} = \theta_i - \eta \nabla \sum_j E_j(\theta)$$

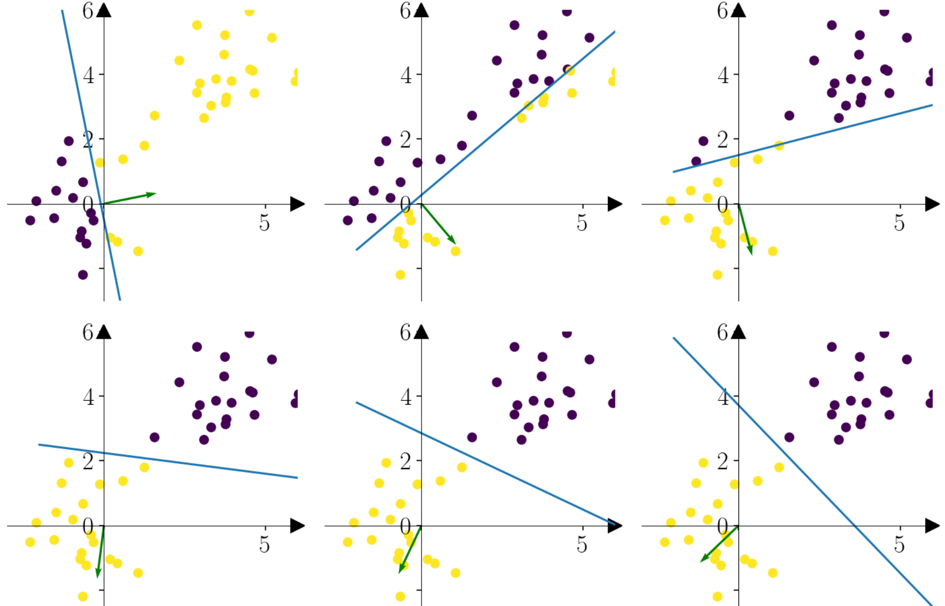
- The error function includes a sum over all data points.
- If we use all data points for the computation of the gradient (batch methods) there would be better ways of doing that than gradient descent.
- Furthermore, the size of the data set often would make it very expensive to use all data points.
- However what we usually do when training neural networks is online learning.
- This means we use only one sample or a subset of samples j (mini-batch) at a time.

- ...

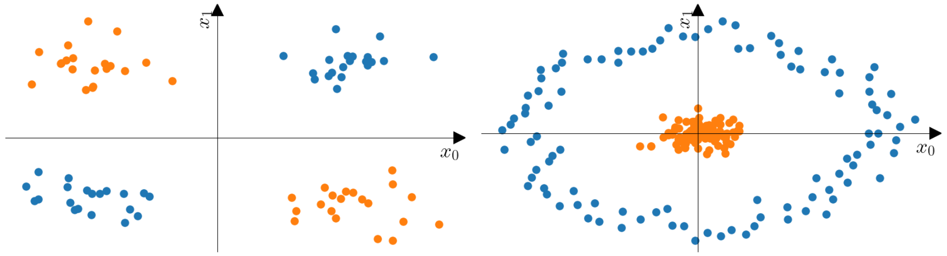
- How to choose samples? $\rightarrow$ Draw randomly without replacement.
- How many samples?
  $\rightarrow$ In CV often as many as possible (VRAM limiting factor)
  $\rightarrow$ Higher batch size $\rightarrow$ less gradient noise $\rightarrow$ higher learning rate η
- However, gradient noise allows to escape local optima!
  $\rightarrow$ Too big batch sizes possible.


6
4
2
0
5

-
-

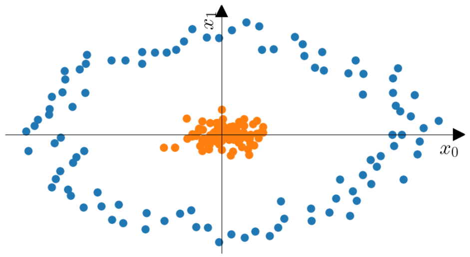

6
4
2
0
5

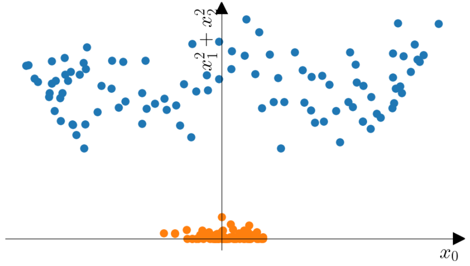
Whats wrong?



-
-

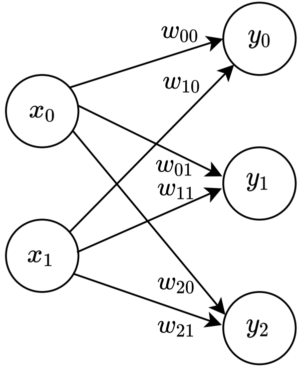
So far we solved this with feature engineering.





- We mapped the images from the highly complex pixel space using e.g. SIFT or HOG into a feature space.
- And afterwards hoped for a good classification result in feature space using e.g. SVM.

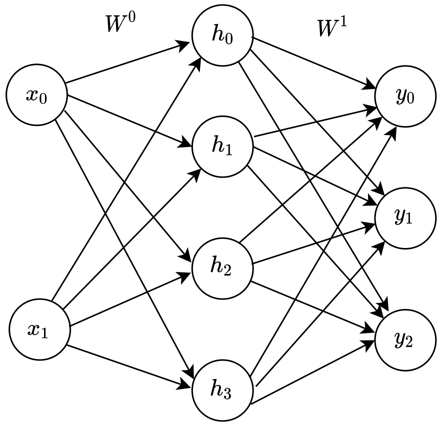
Can we learn these mappings?



$$y = \sigma(Wx)$$
$$\rightarrow y = \sigma(W^1\phi(W^0x))$$

- Let's add another mapping into our Logistic Regression!

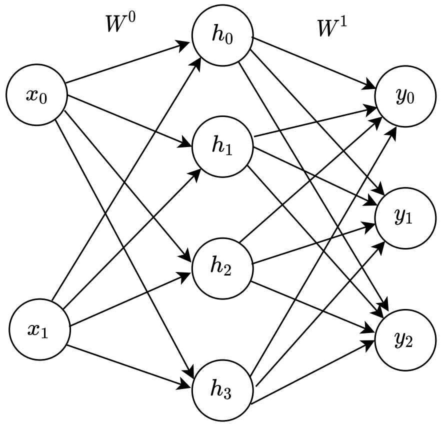
Can we learn these mappings?



$$y = \sigma(W^1\phi(W^0x))$$
$$h = \phi(W^0x)$$

- The vector h is representation in learned feature space.
  $\rightarrow$ It can have any dimensionality.
- ϕ is called the activation function.
  $\rightarrow$ It needs to be non-linear. A purely linear mapping into a new feature space wouldn't help.

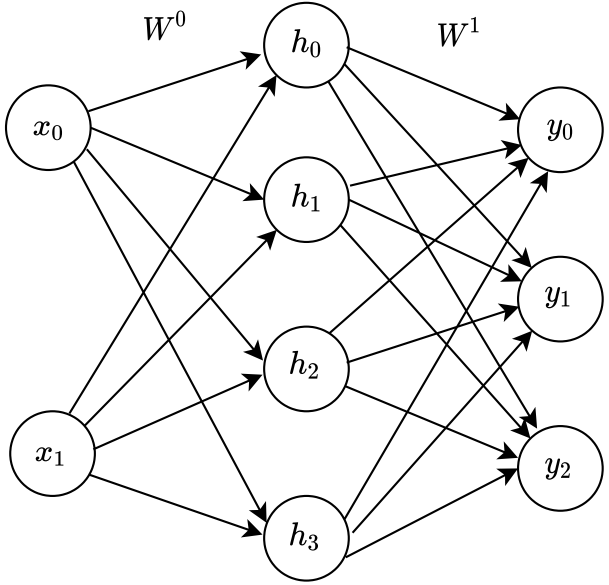
Can we learn these mappings?



$$y = \sigma(W^1 \phi(W^0 x))$$
$$h = \phi(W^0 x)$$

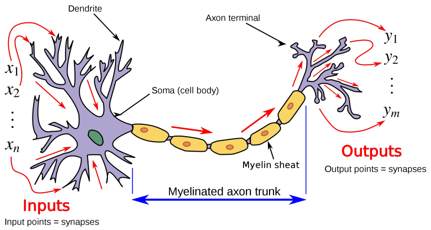
- The vector h is representation in learned feature space.
  $\rightarrow$ It can have any dimensionality.
- ϕ is called the activation function.
  $\rightarrow$ It needs to be non-linear. A purely linear mapping into a new feature space wouldn't help.

Can we learn these mappings? $\rightarrow$ Artificial Neural Network

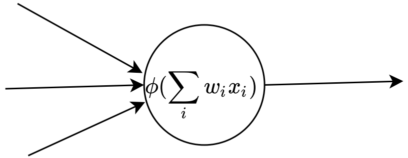


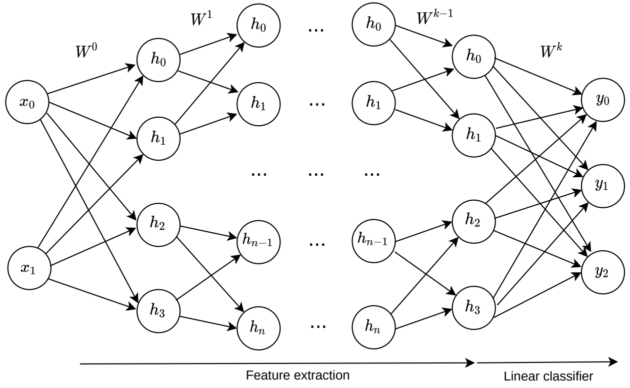
- This is what is called an Artificial Neural Network with one hidden layer.
- It's also sometimes called a Multilayer Perceptron (MLP).

Can we learn these mappings? $\rightarrow$ Artificial Neural Network



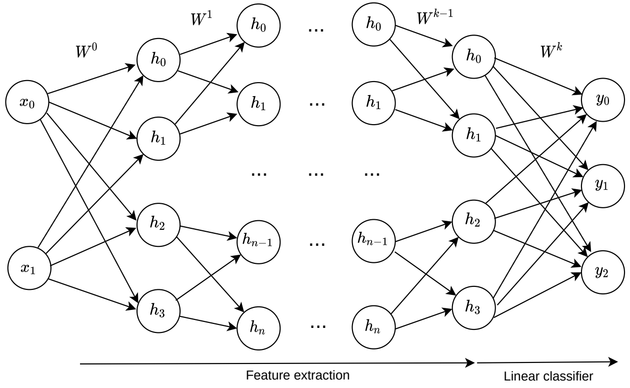
CC Prof. Loc Vu-Quoc





- Adding more hidden layers to this, following the same principle, leads us to what is called deep learning.

But wait. What's the derivative of this?



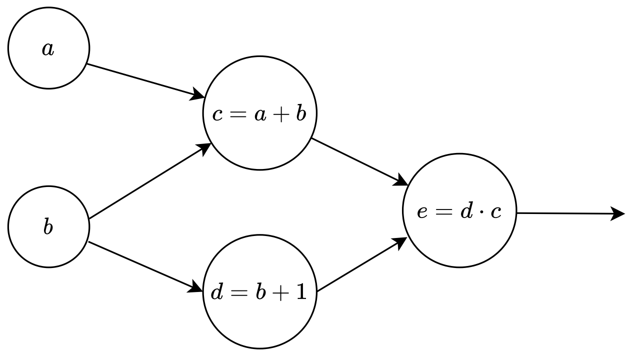
- Manually deriving the derivative? Puhh ...
- Symbolic derivation leads to expression swell.
- Both are restricted to model definitions with closed-form expressions.

But wait. What's the derivative of this?

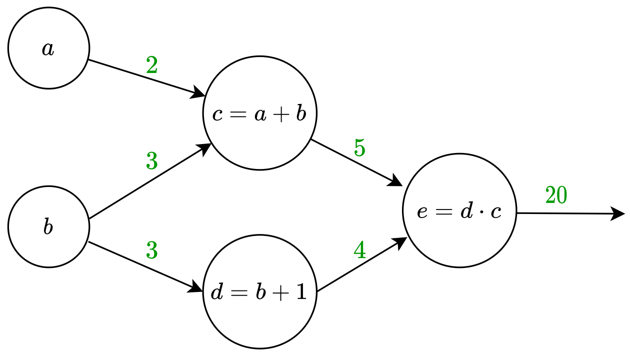
$$\frac{\partial E(w)}{\partial w_i} \approx \frac{E(w + he_i) - E(w)}{h}$$

- e_i is the unit vector in the ith direction and h is a small positive number.
- We could numerically evaluate the gradient for every weight.
  $\rightarrow$ Makes a full evaluation of the network for every weight necessary.

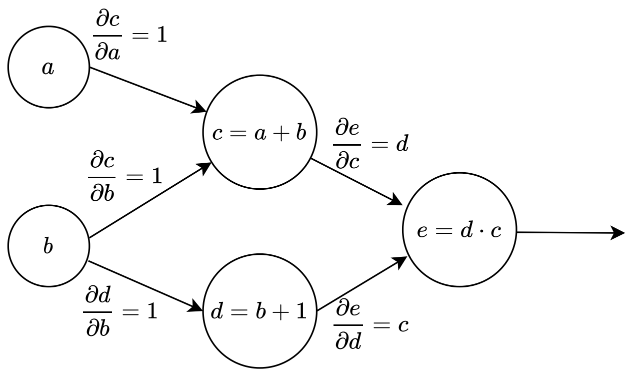
- Fortunately, somebody figured, we could do this using the chain rule!
- The approach was developed many times but
  is widely used in Machine Learning mostly because of
  Learning representations by back-propagating errors.
  David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams; Nature; 1986



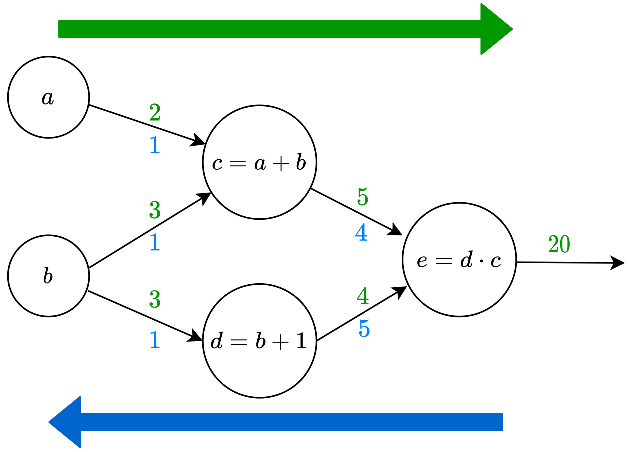
- Let's look at the computational graph for the function $e = (a + b)(b + 1)$



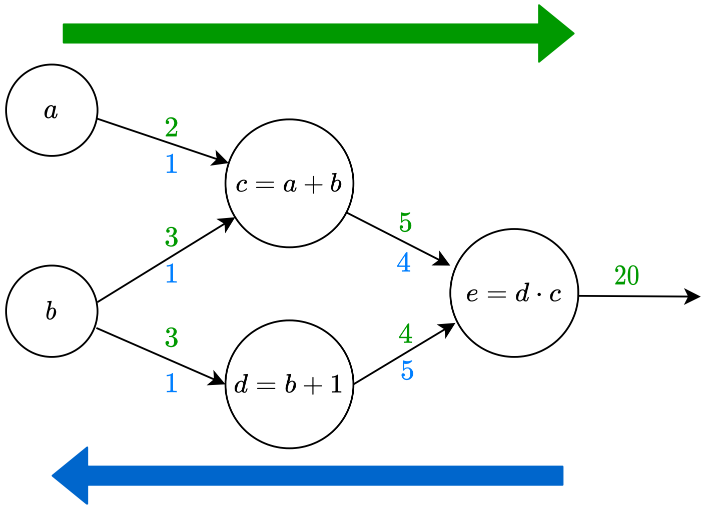
- A forward pass through this graph for $a = 2$ and $b = 3$.



- We can assign the partial derivative to each edge of the graph.

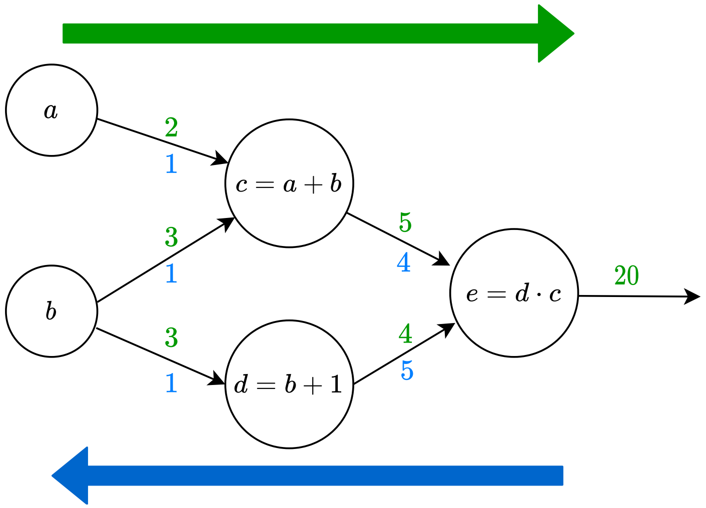


- After a forward pass through the network, we can assign a value to all of the partial derivatives by doing what is called a backward pass.



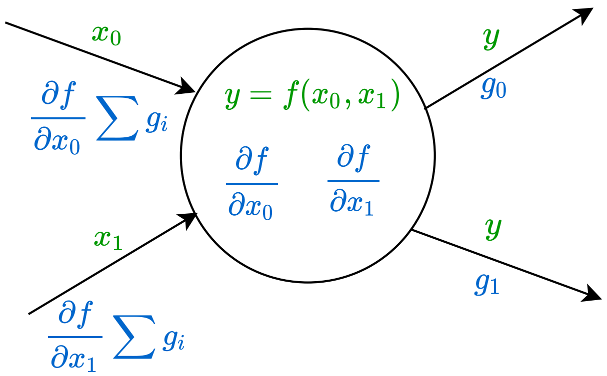
$$\frac{\partial e}{\partial a} = 4 \cdot 1 = \frac{\partial e(c(b))}{\partial c} \frac{\partial c(b)}{\partial b}$$

- If we multiply the values of the partial derivatives of the nodes from output to parameter, we get the partial derivative of the full network with respect to this parameter.
- Which is nothing else than the application of the chain rule.

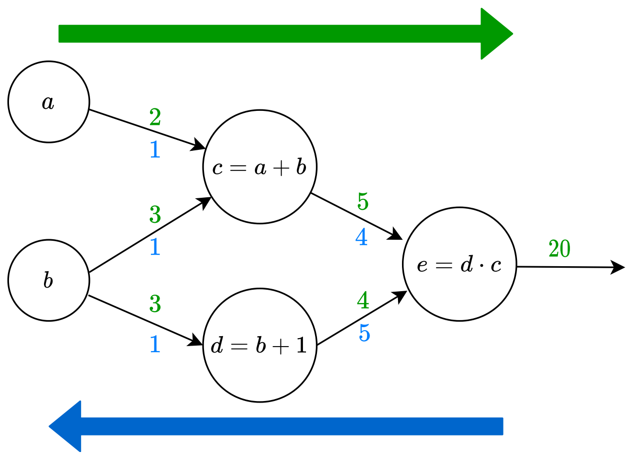


$$\frac{\partial e}{\partial b} = 4 \cdot 1 + 5 \cdot 1$$
$$= \frac{\partial e(c(b))}{\partial c}\frac{\partial c(b)}{\partial b} + \frac{\partial e(d(b))}{\partial d}\frac{\partial d(b)}{\partial b}$$

- If there are multiple paths from output to parameter, we have to sum up all the derivatives at every node before propagating the further.
- This also corresponds to the chain rule in the multi variate case.

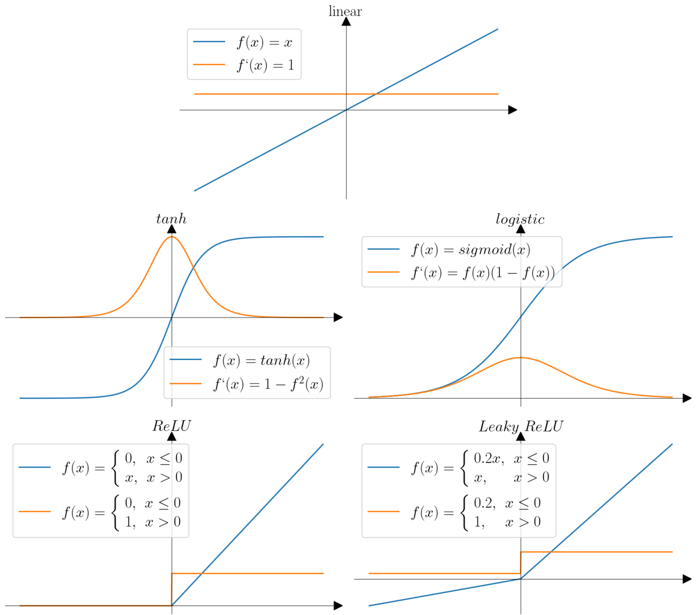


- Every node needs to implement a function and the partial derivatives of that function with respect to its inputs.
  $\rightarrow$ The values of the partial derivatives of the network with respect to all parameters for a given input, can be computed with one forward and one backward pass.



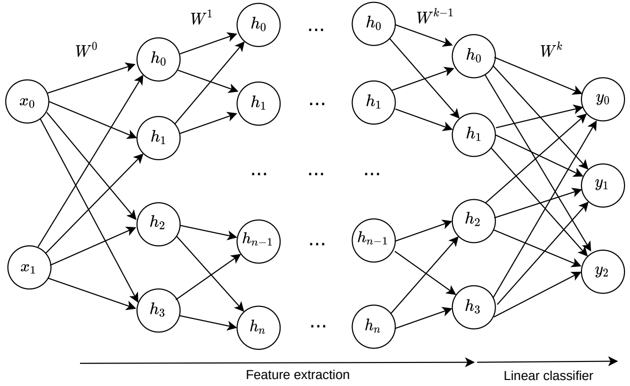
- Activations (node output) of all layers have to be kept in memory for the backward path!
  $\rightarrow$ High memory consumption of the network during training.
  $\rightarrow$ (Yes, Backprop is dynamic programming.)
- Add-nodes distribute gradient equally.
- Multiply-nodes backpropagate their inputs as gradients.
- What does that mean e.g. for the Wx operation?
  $\rightarrow$ Big input, big gradient on the weights!
  $\rightarrow$ Preprocessing of input data matters for gradient flow!
  $\rightarrow$ To understand and monitor gradient flow is crucial for successful training of neural networks.

Activation function

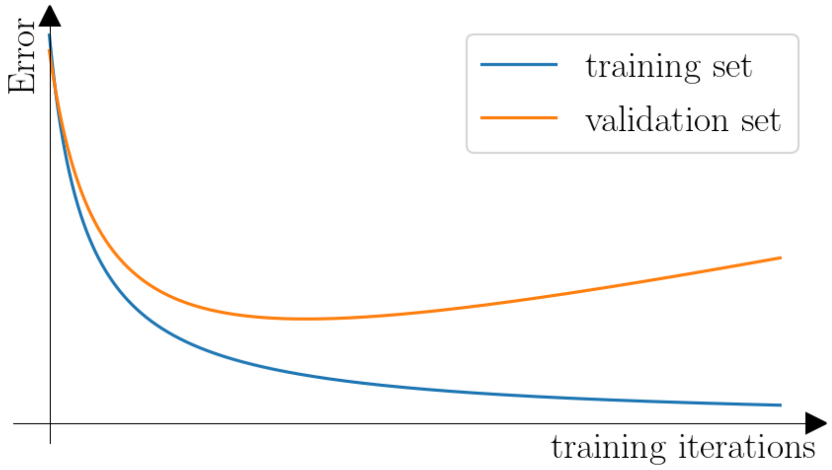


- Let's have a look at possible activation functions.

"With great power comes great overfitting" (Joseph Redmon)



- Deep neural networks have millions of parameters.
- They can learn to fit almost everything. But that's often not what you want.



- When fitting a model with a gradient descent method, we should always look at the error on training and validation sets over training iteration.
- High bias
  → Increase model complexity
- High variance
  → More data and/or regularization
- In deep learning it's less of a trade-off between bias and variance.
  → We can increase model complexity accompanied with more data/regularization until bias is close to zero, without increasing variance much.

- Early stopping
- Weight regularization
- Dropout
- Data augmentation
- Batch normalization