Introduction to Machine Learning

Lecture 15
Deep Learning II
Backpropagation

Goker Erdogan
26 – 30 November 2018
Pontificia Universidad Javeriana

Universal approximation property

- Neural networks are universal approximators
  - 2 layer network can approximate any continuous function to arbitrary accuracy
    - Given that the network has enough units

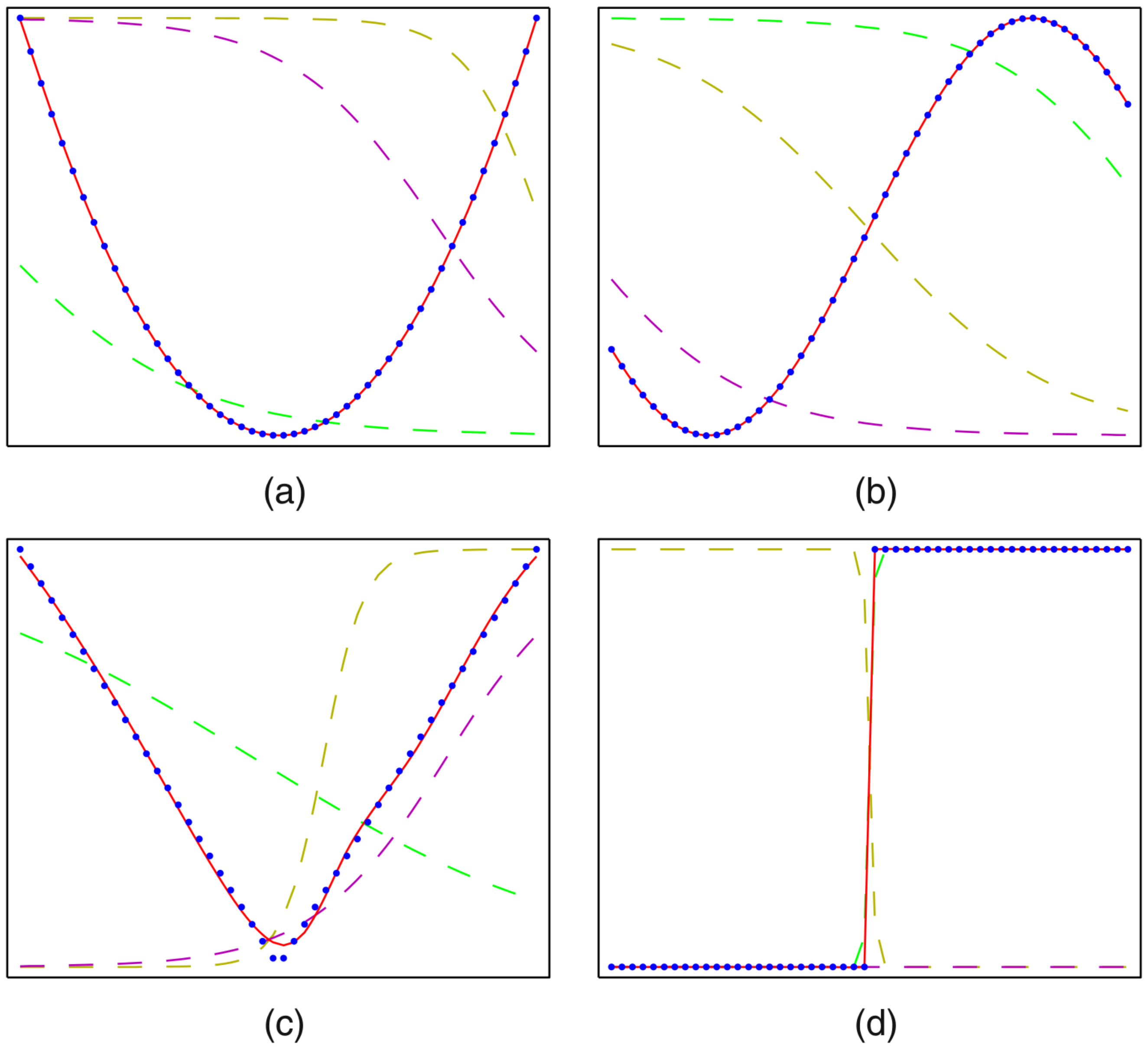


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with '$\tanh$' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

Universal approximation property

- Neural networks are universal approximators
  - 2 layer network can approximate any continuous function to arbitrary accuracy
    - Given that the network has enough units

- Note this does not tell you whether
  - Easy/hard to train the network to approximate function well
  - The network will generalize well to test data

- May need a large number of units to approximate some function
  - Adding more layers may reduce this number significantly

Training a neural network

- Given a training sample {x, t}
  - A neural network defines a function x → y
    - With parameters w

- Define an error function
  - e.g. regression: sum of squares error

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

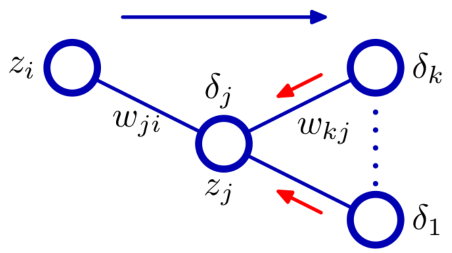
- Find the derivative of E with respect to parameters w
  - Use gradient-based optimization techniques to learn w

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \nabla_{\boldsymbol{w}} E(\boldsymbol{w})$$

(Error) Backpropagation

- Forward propagation
  - Calculating the output and error function given inputs
  - Values are propagated forward in the network

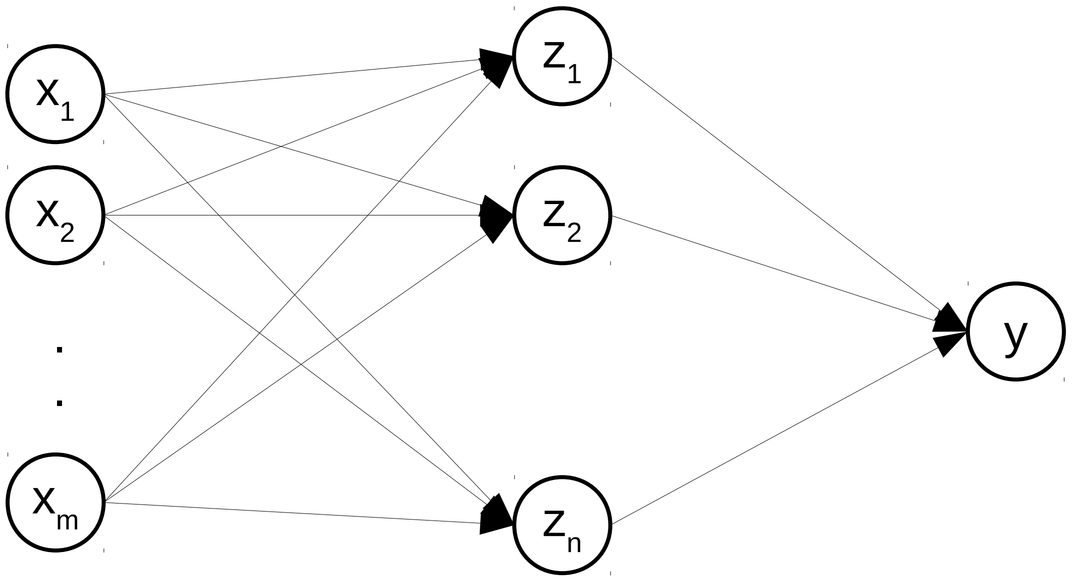
- Backpropagation
  - An efficient way to calculate the derivative $\nabla_{\boldsymbol{w}} E(\boldsymbol{w})$
    - Just an application of the chain rule of calculus
  - Calculate errors and backpropagate them



Chain rule of calculus

$$\begin{tikzcd} x \arrow[r] & z \arrow[r] & y \end{tikzcd}$$

$$\frac{dy}{dx} = \frac{dy}{dz}\frac{dz}{dx}$$



$$\frac{\partial y}{\partial x_i} = \sum_j \frac{\partial y}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$

Backprop algorithm

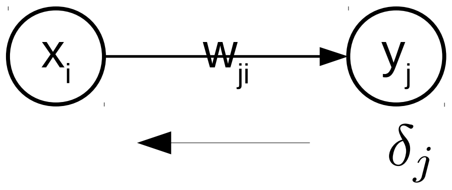
- Consider a single layer (linear) network

$$y_j = \sum_i w_{ji} x_i$$

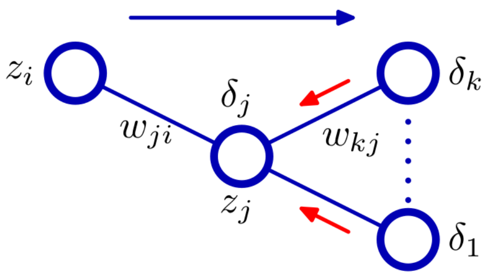
  - With error function

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_j (y_j - t_j)^2$$

- Then the derivative is



$$\frac{\partial E}{w_{ji}} = \underbrace{(y_j - t_j)}_{\substack{\delta_j \\ \text{error}}} \underbrace{x_i}_{\text{input}}$$



Notation:
i, j, k indices refer to units in different layers

- In the general case,

$$a_j = \sum_i w_{ji} z_i \qquad z_j = h(a_j)$$

$$\frac{\partial E}{\partial w_{ji}} = \underbrace{\frac{\partial E}{\partial a_j}}_{\delta_j} \underbrace{\frac{\partial a_j}{\partial w_{ji}}}_{z_i}$$

$$\delta_j = \sum_k \underbrace{\frac{\partial E}{\partial a_k}}_{\delta_k} \frac{\partial a_k}{\partial a_j} \longrightarrow \delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

Backprop example

- 2-layer network with tanh activation function and sum-of-squares error

$$\begin{aligned} a_j &= \sum_{i=0}^{D} w_{ji}^{(1)} x_i \\ z_j &= \tanh(a_j) \\ y_k &= \sum_{j=0}^{M} w_{kj}^{(2)} z_j. \end{aligned}$$

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

- Backprop equations

$$\delta_k = y_k - t_k. \qquad \delta_j = (1 - z_j^2)\sum_{k=1}^{K} w_{kj}\delta_k.$$

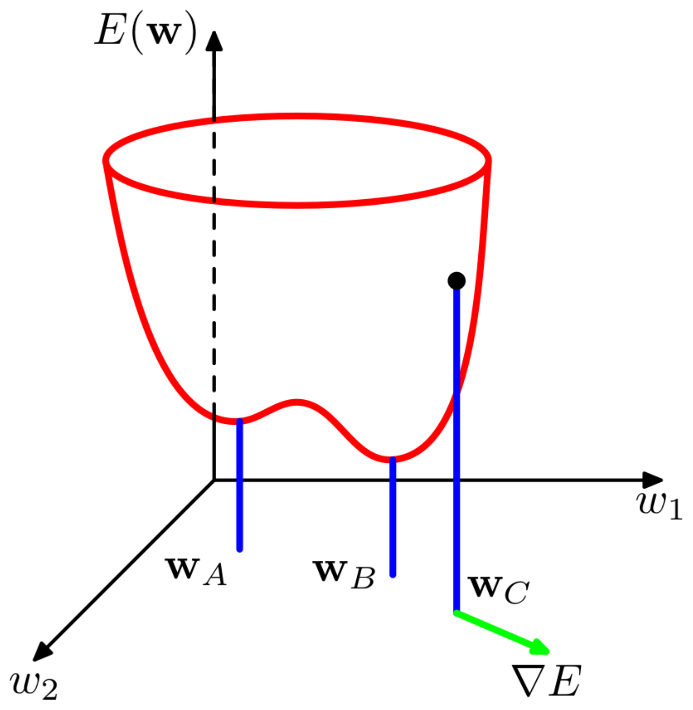
Note:

$$h'(a) = 1 - h(a)^2.$$

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i, \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j.$$

Training a neural network

- For most neural networks, the optimization problem is non-convex
  - Multiple minima
  - But in practice, gradient-based techniques work well
    - Local minima seem to give good solutions



Backpropagation

- General technique for calculating derivatives
  - Efficient because it re-uses computations

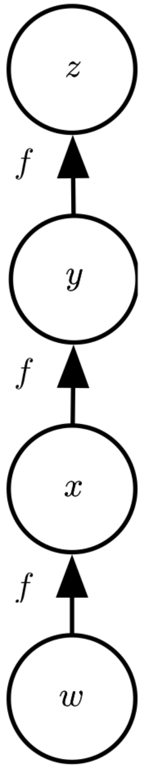


Figure 6.9: A computational graph that results in repeated subexpressions when computing the gradient. Let $w \in \mathbb{R}$ be the input to the graph. We use the same function $f : \mathbb{R} \to \mathbb{R}$ as the operation that we apply at every step of a chain: $x = f(w)$, $y = f(x)$, $z = f(y)$. To compute $\frac{\partial z}{\partial w}$, we apply equation 6.44 and obtain:

$$\frac{\partial z}{\partial w} \tag{6.50}$$

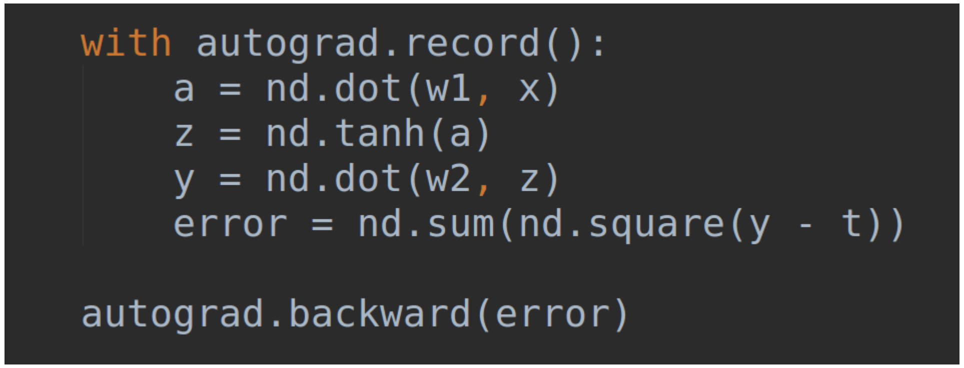
$$= \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \frac{\partial x}{\partial w} \tag{6.51}$$

$$= f'(y) f'(x) f'(w) \tag{6.52}$$

$$= f'(f(f(w))) f'(f(w)) f'(w). \tag{6.53}$$

Backpropagation

- General technique for calculating derivatives
  - Efficient because it re-uses computations
    - If memory is an issue, we can choose to re-calculate some
  - Roughly the same computation cost as forward propagation
  - Can be applied to any function (not only neural networks)
  - Known as reverse mode differentiation in automatic differentiation field
  - Good libraries to take derivatives of any (differentiable) function
    - e.g., Tensorflow, Theano, Torch, MXNet



```
with autograd.record():
    a = nd.dot(w1, x)
    z = nd.tanh(a)
    y = nd.dot(w2, z)
    error = nd.sum(nd.square(y - t))

autograd.backward(error)
```

Summary

- Universal approximation property
- Backpropagation
  - Chain rule of calculus
  - Backprop equations
  - Backprop example
  - Non-convex problem

- Exercises
  - Derive the backprop equations for a 2-layer network with ReLU activations

References

[1] Goodfellow I., Bengio Y., Courville A. Deep Learning.
https://www.deeplearningbook.org/

[2] Bishop C. Pattern Recognition and Machine Learning. Chapter 5.