GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 6

Machine learning

Neural networks

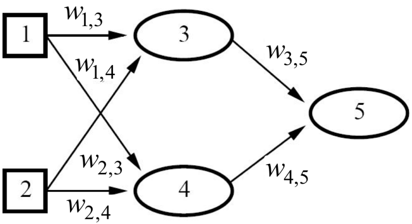
Part 2

Aleksandra Pizurica

Ghent University

Spring 2024

Feed-forward networks - example



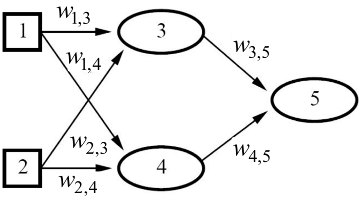
Feed-forward network $=$ a parameterized family of nonlinear functions:

$$
\begin{aligned}
a_5 &= g(w_{3,5}a_3 + w_{4,5}a_4) \\
&= g\Big(w_{3,5}g(w_{1,3}x_1 + w_{2,3}x_2) + w_{4,5}g(w_{1,4}x_1 + w_{2,4}x_2)\Big)
\end{aligned}
$$

Adjusting weights changes the function: do learning this way!

Weight matrix

(For simplicity, we assume no bias)



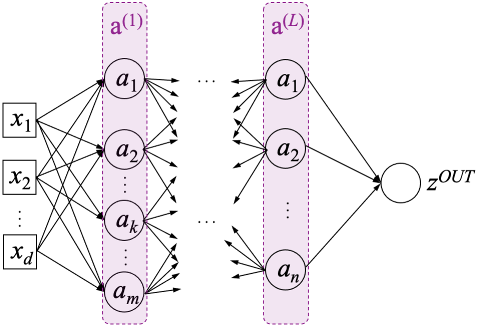
$$a_5 = g\left(\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix} \begin{bmatrix} a_3 \\ a_4 \end{bmatrix}\right) = g\Bigg(\underbrace{\begin{bmatrix} w_{3,5} & w_{4,5} \end{bmatrix}}_{\mathbf{W}^{(2)}} g\Bigg(\underbrace{\begin{bmatrix} w_{1,3} & w_{2,3} \\ w_{1,4} & w_{2,4} \end{bmatrix}}_{\mathbf{W}^{(1)}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\Bigg)\Bigg)$$

$$= g\left(\mathbf{W}^{(2)} g\left(\mathbf{W}^{(1)}\mathbf{x}\right)\right)$$

$\mathbf{W}^{(l)}$ is the **weight matrix** in the l-th layer. Its rows are the weight vectors. Omitting the layer index: $\mathbf{W}(j,:) = \mathbf{w}_j{}^\top = [w_{0,j} \ldots w_{n,j}]$.

Matrix notation for multilayer neural networks

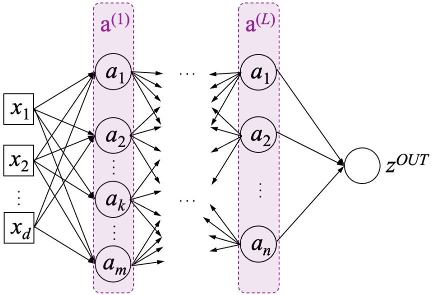
For now still assume **no bias** (What should change in the equations otherwise?)



$$\mathbf{a}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x})$$
$$\mathbf{a}^{(2)} = g(\mathbf{W}^{(2)}\mathbf{a}^{(1)})$$
$$\vdots$$
$$\mathbf{a}^{(L-1)} = g(\mathbf{W}^{(L-1)}\mathbf{a}^{(L-2)})$$
$$z^{OUT} = \mathbf{W}^{(L)}\mathbf{a}^{(L-1)}$$

Regression: $h_{\mathbf{w}}(\mathbf{x}) = z^{OUT}$; Classification: $h_{\mathbf{w}}(\mathbf{x}) = Threshold(z^{OUT})$
(or feed the output scores to sigmoid or to softmax for multiclass classification)

Matrix notation for multilayer neural networks

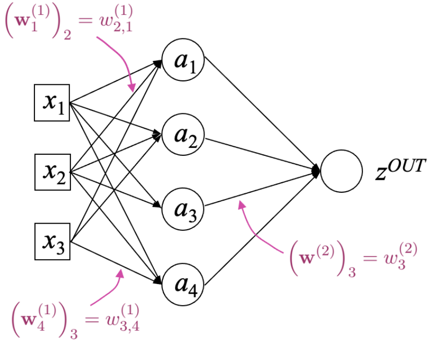


For m neurons in layer i with d inputs:

$$\mathbf{W}^{(i)} = \begin{bmatrix} \mathbf{w}_1^{(i)\top} \\ \vdots \\ \mathbf{w}_m^{(i)\top} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{(i)} \ldots w_{d,1}^{(i)} \\ \vdots \\ w_{1,m}^{(i)} \ldots w_{d,m}^{(i)} \end{bmatrix}$$

$$z^{OUT} = \mathbf{W}^{(n)} g\left(\mathbf{W}^{(n-1)} \ldots g\left(\mathbf{W}^{(1)}\mathbf{x}\right)\right)$$

Matrix notation for multilayer neural networks



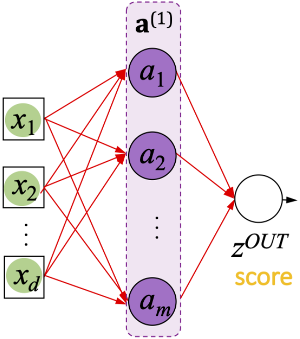
For the neural network on the left:

$$\mathbf{W}^{(1)} = \begin{bmatrix} \mathbf{w}_1^{(1)\top} \\ \vdots \\ \mathbf{w}_4^{(1)\top} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{(1)} & w_{2,1}^{(1)} & w_{3,1}^{(1)} \\ \vdots & & \\ w_{1,4}^{(1)} & w_{2,4}^{(1)} & w_{3,4}^{(1)} \end{bmatrix}$$

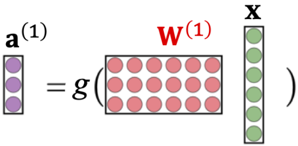
$$\mathbf{W}^{(2)} = \left[\mathbf{w}^{(2)\top}\right] = \left[w_1^{(2)} \quad w_2^{(2)} \quad w_3^{(2)} \quad w_4^{(2)}\right]$$

$$\underbrace{z^{OUT}}_{\text{score}} = \mathbf{w}^{(2)} \cdot g\left(\mathbf{W}^{(1)}\mathbf{x}\right) = \mathbf{w}^{(2)\top} g\left(\mathbf{W}^{(1)}\mathbf{x}\right)$$

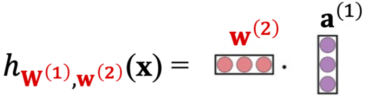
Two-layer regression neural network



Intermediate problems (learned features):



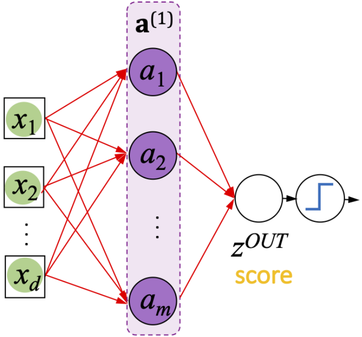
Predictor:



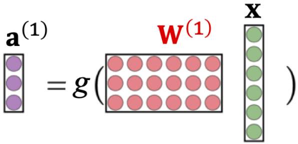
Hypothesis class:

$$\mathcal{H} = \{h_{\mathbf{W}^{(1)},\mathbf{w}^{(2)}} : \mathbf{W}^{(1)} \in \mathbb{R}^{m\times d}, \mathbf{w}^{(2)} \in \mathbb{R}^{m}\}$$

Two-layer classification neural network



Intermediate problems (learned features):



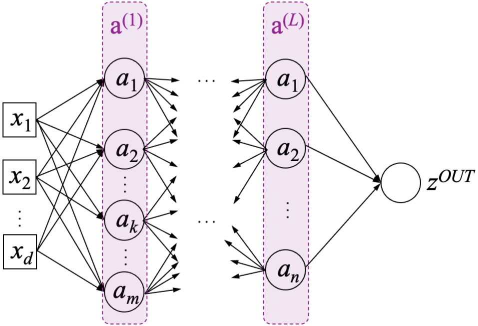
Predictor:

Hypothesis class:

$$\mathcal{H} = \{h_{\mathbf{W}^{(1)},\mathbf{w}^{(2)}} : \mathbf{W}^{(1)} \in \mathbb{R}^{m \times d}, \mathbf{w}^{(2)} \in \mathbb{R}^{m}\}$$

Compact matrix notation for multilayer neural networks

We can use the same compact notation when bias is not omitted

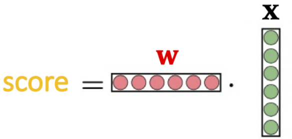


$$\mathbf{a}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x})$$
$$\mathbf{a}^{(2)} = g(\mathbf{W}^{(2)}\mathbf{a}^{(1)})$$
$$\vdots$$
$$\mathbf{a}^{(L-1)} = g(\mathbf{W}^{(L-1)}\mathbf{a}^{(L-2)})$$
$$z^{OUT} = \mathbf{W}^{(L)}\mathbf{a}^{(L-1)}$$

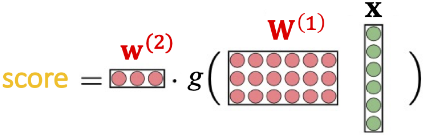
- We stipulate: each unit has an extra input from a dummy unit that is fixed to $+1$ and a weight $w_{0,j}$ for that input

Multilayer neural networks in matrix notation

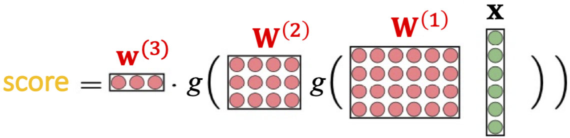
1-layer neural network:



2-layer neural network:

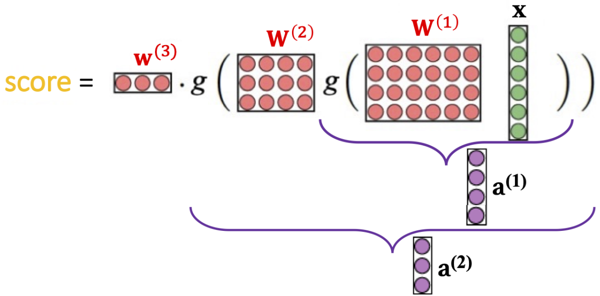


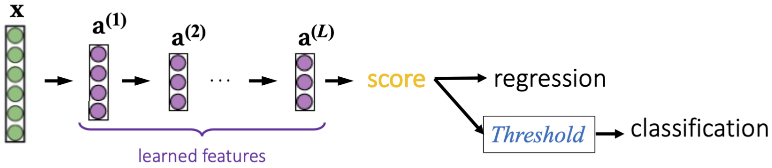
3-layer neural network:



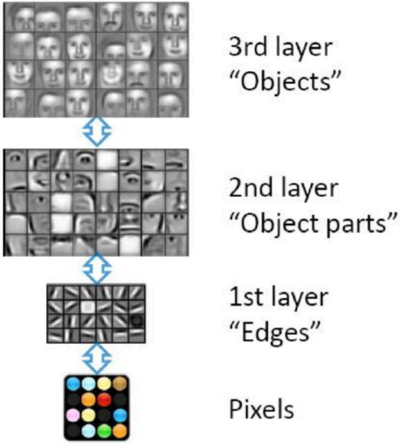
Slide adapted from: M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

Multilayer neural networks in matrix notation





Layers represent multiple levels of abstractions



Optimization in neural networks: A motivating example

Consider regression with a four-layer neural network.

Loss on one example:

$$Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{w}^{(4)}) = (\mathbf{w}^{(4)} \cdot g(\mathbf{W}^{(3)} g(\mathbf{W}^{(2)} g(\mathbf{W}^{(1)} \mathbf{x}))) - y)^2$$

(Stochastic) gradient descent:

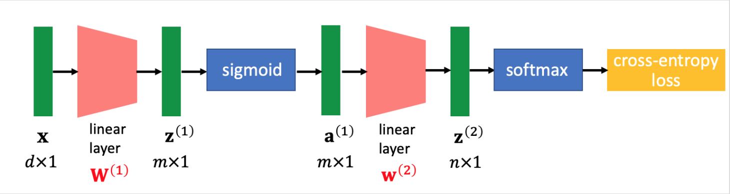
$$\begin{aligned}
\mathbf{W}^{(1)} &\leftarrow \mathbf{W}^{(1)} - \alpha \nabla_{\mathbf{W}^{(1)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{w}^{(4)}) \\
\mathbf{W}^{(2)} &\leftarrow \mathbf{W}^{(2)} - \alpha \nabla_{\mathbf{W}^{(2)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{w}^{(4)}) \\
\mathbf{W}^{(3)} &\leftarrow \mathbf{W}^{(3)} - \alpha \nabla_{\mathbf{W}^{(3)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{w}^{(4)}) \\
\mathbf{w}^{(4)} &\leftarrow \mathbf{W}^{(3)} - \alpha \nabla_{\mathbf{w}^{(4)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}, \mathbf{w}^{(4)})
\end{aligned}$$

We learned to compute the gradients in the inner layers by backpropagation.

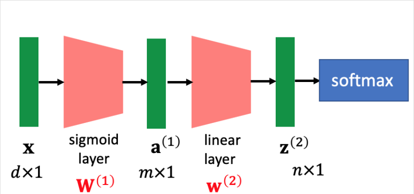
Still, this seems very complex!

Easy with automatic differentiation tools that use computation graphs.

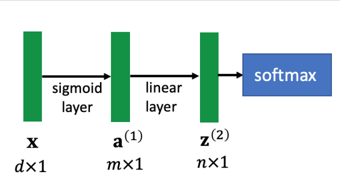
Digression: How do we typically represent deep neural nets



Simpler:

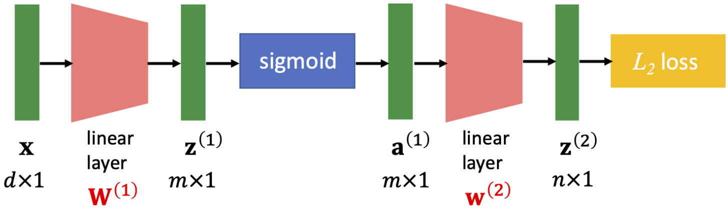


or even simpler:



Sergey Levine: Backpropagation - Designing, Visualizing and Understanding Deep Neural Networks.

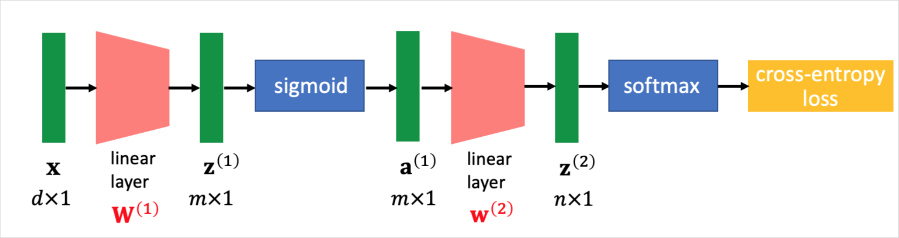
Gradient of the loss function



$$\nabla_{\mathbf{w}} Loss = \left(\frac{\partial Loss}{\partial \mathbf{w}}\right)^{\top}$$

$$\frac{\partial Loss}{\partial \mathbf{w}^{(2)}} \in \mathbb{R}^{1\times n} \; ; \qquad \frac{\partial Loss}{\partial \mathbf{W}^{(1)}} \in \mathbb{R}^{m\times d}$$

Backpropagation



$$\frac{\partial Loss}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}} \underbrace{\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \underbrace{\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}} \underbrace{\frac{\partial Loss}{\partial \mathbf{z}^{(2)}}}_{\delta_{init}}}_{\text{compute first: } \delta}}_{\text{update: new } \delta}$$

Why this recursion?

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

$\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}}, \frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}, \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}$ are Jacobian matrices

Suppose $m = n$ ($\mathbf{a}^{(i)}$, $\mathbf{z}^{(i)}$ are of size n)

Both $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}}, \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}$ are $n \times n$

Computing $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}}$ is $O(n^3)$!

(AlexNet has layers with $n = 4096$)

Make "cheap" computation in each step

Initialize: $\frac{\partial Loss}{\partial \mathbf{z}^{(2)}} = \delta_{init}$

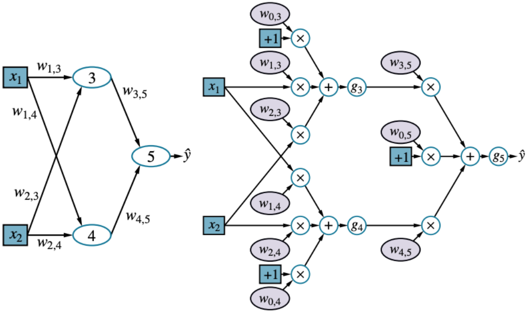
compute: $\frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{a}^{(1)}} \delta_{init} = \delta$

$$\frac{\partial Loss}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}} \underbrace{\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \delta}_{O(n^2)}$$

compute: $\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{z}^{(1)}} \delta = \delta_{new}$

$$\frac{\partial Loss}{\partial \mathbf{W}^{(1)}} = \underbrace{\frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(1)}} \delta_{new}}_{O(n^2)}$$

Computation graphs

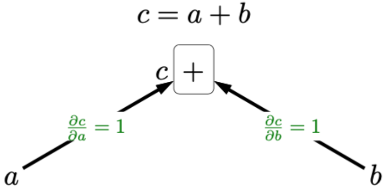


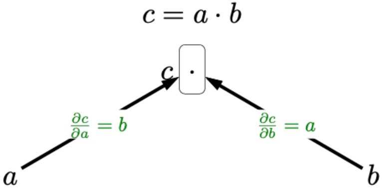
Definition (Computation graph)

A directed acyclic graph whose root node represents the final mathematical expression and each node represents intermediate subexpressions.

- Automatically compute gradients (how TensorFlow and PyTorch work)
- Gain insight into modular structure of gradient computations

Computation graphs concepts: Functions as boxes





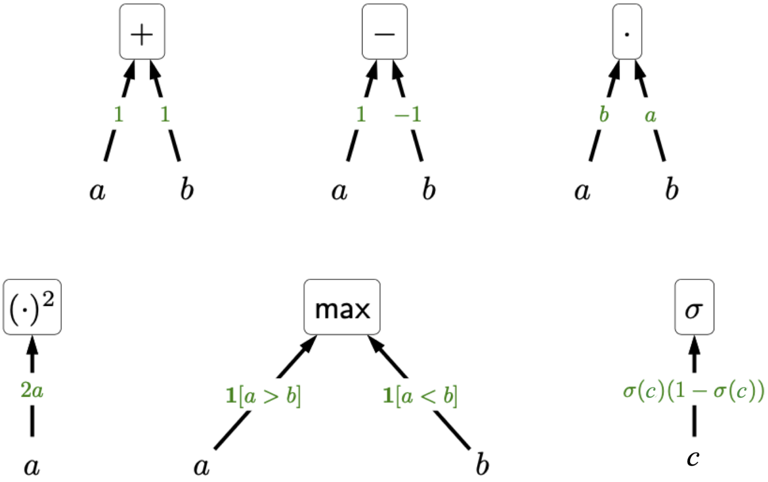
$$(a + \epsilon) + b = c + 1\epsilon$$
$$a + (b + \epsilon) = c + 1\epsilon$$

$$(a + \epsilon)b = c + b\epsilon$$
$$a(b + \epsilon) = c + a\epsilon$$

Gradients: how much does c change if a or b changes?

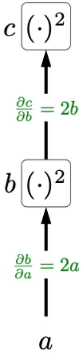
M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

Basic building blocks of computation graphs



σ denotes sigmoid (logistic) function. M. Charikar & Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

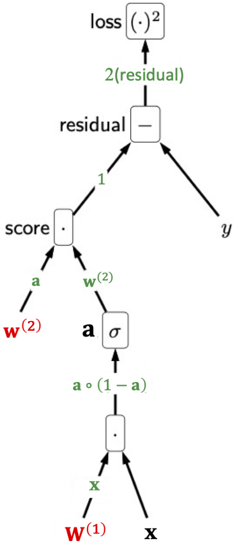
Function composition



$$\frac{\partial c}{\partial a} = \frac{\partial c}{\partial b}\frac{\partial b}{\partial a} = (2b)(2a) = (2a^2)(2a) = 4a^3$$

M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

Two-layer neural networks

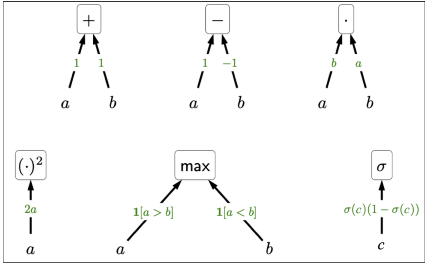


$$Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{w}^{(2)}) = (\mathbf{w}^{(2)} \cdot g(\mathbf{W}^{(1)}\mathbf{x}) - y)^2$$

Let $g(x) = \sigma(x)$ (sigmoid activation function)

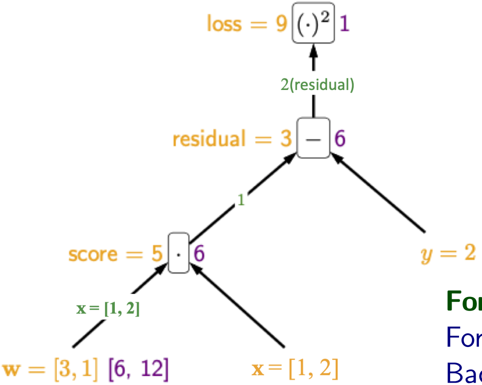
$$\nabla_{\mathbf{w}^{(2)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{w}^{(2)}) = 2(\text{residual})\mathbf{a}$$

$$\nabla_{\mathbf{W}^{(1)}} Loss(\mathbf{x}, y, \mathbf{W}^{(1)}, \mathbf{w}^{(2)}) = 2(\text{residual})\mathbf{w}^{(2)} \circ \mathbf{a} \circ (1 - \mathbf{a})\mathbf{x}^\top$$



Adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

A simple backpropagation example



$$Loss(x, y, \mathbf{w}) = (\mathbf{w} \cdot \mathbf{x} - y)^2$$

$\mathbf{w} = [3, 1], \quad \mathbf{x} = [1, 2], y = 2$

backpropagation

$\nabla_{\mathbf{w}} \text{Loss}(\mathbf{x}, y, \mathbf{w}) = [6, 12]$

Forward/backward values:

Forward: f_i is value for subexpression rooted at i

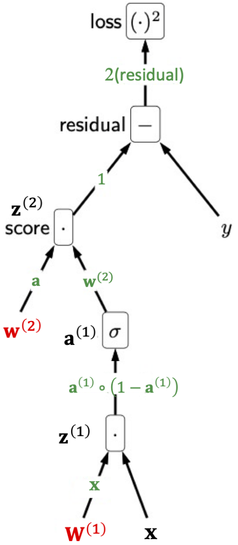
Backward: $b_i = \frac{\partial Loss}{\partial f_i}$ is how f_i influences loss

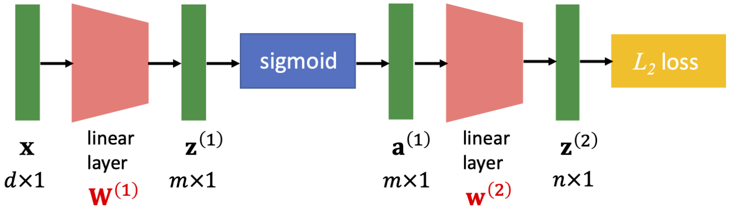
Backpropagation:

1. Forward pass: compute each f_i (from leaves to root)
2. Backward pass: compute each b_i (from root to leaves)

Adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

Backpropagation in neural networks explained





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

Easily from the computation graph:

$$2(\text{residual})\mathbf{w}^{(2)} \circ \mathbf{a}^{(1)} \circ (1 - \mathbf{a}^{(1)})\mathbf{x}^\top$$

Cross-entropy loss

In deep learning, commonly we talk about minimizing **cross-entropy** loss

- Cross-entropy $H(P,Q)$ is a measure of dissimilarity
  between the two distributions P and Q
- General definition:

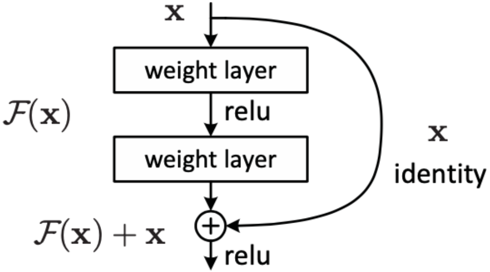
$$H(P,Q) = \mathbb{E}_{z\sim P(z)}[-\log Q(z)] = -\int P(\mathbf{z}) \log Q(\mathbf{z}) d\mathbf{z}$$

- Typically: P is the true distribution over the training examples $P^*(\mathbf{x}, y)$, and Q is the predictive hypothesis $P(y|\mathbf{x}, \mathbf{w})$
  - But we don't know $P^*(\mathbf{x}, y)$. We have access to its samples though!
  - So, approximate the expectation by the sum over the samples
  - Practical approach:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} -\sum_{j=1}^{N} \log P(y^{(j)}|\mathbf{x}^{(j)}, \mathbf{w}) = \arg\max_{\mathbf{w}} \sum_{j=1}^{N} \log P(y^{(j)}|\mathbf{x}^{(j)}, \mathbf{w})$$

Residual neural networks

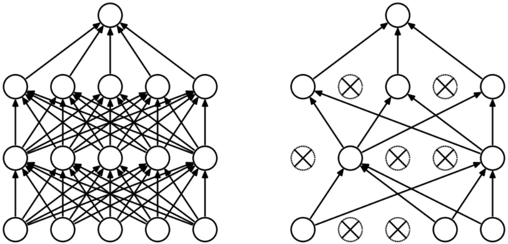
- A popular approach to building very deep neural networks
- Instead of learning the desired mapping $h(\mathbf{x})$, the stacked nonlinear layers fit the residual $\mathcal{F}(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$. Hence, the original mapping recast to $\mathcal{F}(\mathbf{x}) + \mathbf{x}$
- It is easier to optimize the residual mapping than to optimize the original, unreferenced mapping
  - Think if an identity mapping were optimal, easier to push residual to zero than to fit identity mapping by a stack of nonlinear layers



Regularization in deep neural networks

Some common regularization approaches in deep learning include

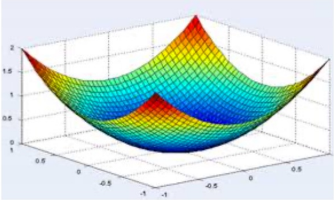
- Weight decay: add a penalty $\lambda \sum_{i,j} w_{i,j}^2$ to the loss function
  - Not straightforward to interpret the effect of weight decay in neural network
  - Common to use λ near 10^{-4}
- Dropout: deactivate a random chosen subset of units in each step of training



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

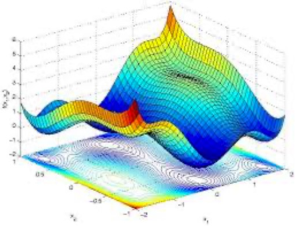
Does stochastic gradient descent (SGD) work for neural networks?

Linear predictors



(convex)

Neural networks



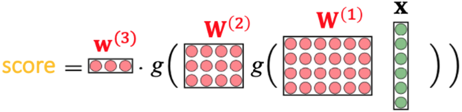
(non-convex)

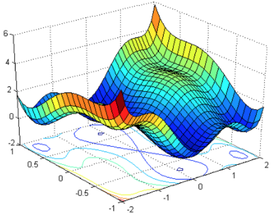
- For neural networks, optimization is hard
- In practice, SGD can work for neural nets much better than the theory predicts

  The gap between theory and practice not well understood yet!

Adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).

How to train neural networks





- Careful initialization (random noise, pre-training)
- Overparameterization (more hidden units than needed)
- Adaptive step sizes (AdaGrad, Adam)
- **Don't let gradients vanish or explode!**

Adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford).