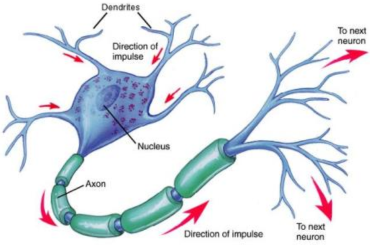
Topic 10: NEURAL NETWORKS

STAT 37710/CAAM 37710/CMSC 35400 Machine Learning

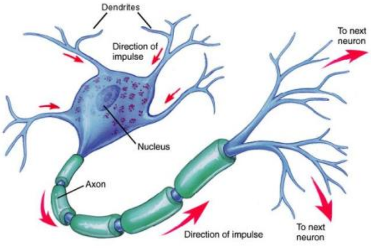
Risi Kondor, The University of Chicago

Intro to neural networks



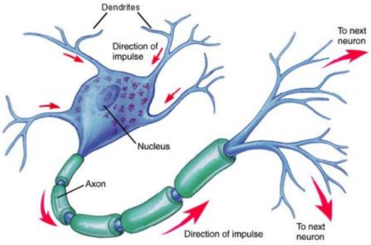
- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^{4}$ others.

Intro to neural networks



- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^4$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

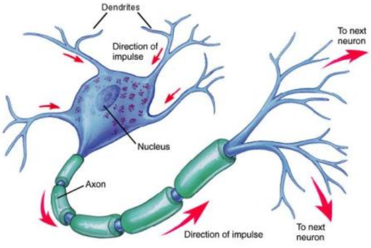
Intro to neural networks



- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^{4}$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.

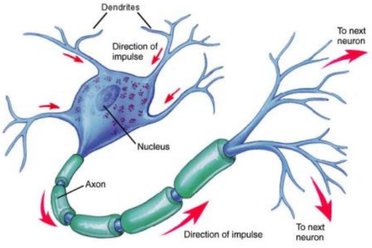
Intro to neural networks



- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^4$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.
- Information storage is the nervous system is “distributed”.

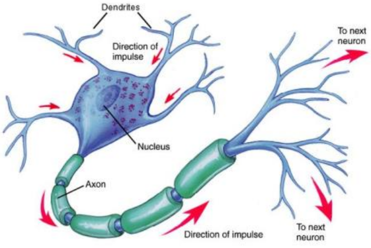
Intro to neural networks



- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^4$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.
- Information storage is the nervous system is "distributed".
- The response time of the brain is quite fast, so the "depth" of the network can't be very great. (clear layer by layer organization in the visual system).

Intro to neural networks

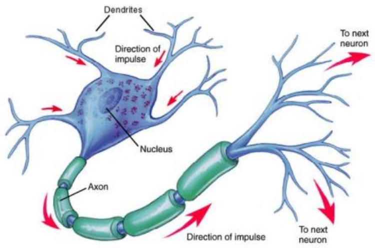


- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^{4}$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.
- Information storage is the nervous system is “distributed”.
- The response time of the brain is quite fast, so the “depth” of the network can’t be very great. (clear layer by layer organization in the visual system).

IDEA: Humans seem to be okay at learning, so why not try to replicate this in a computer?

Intro to neural networks

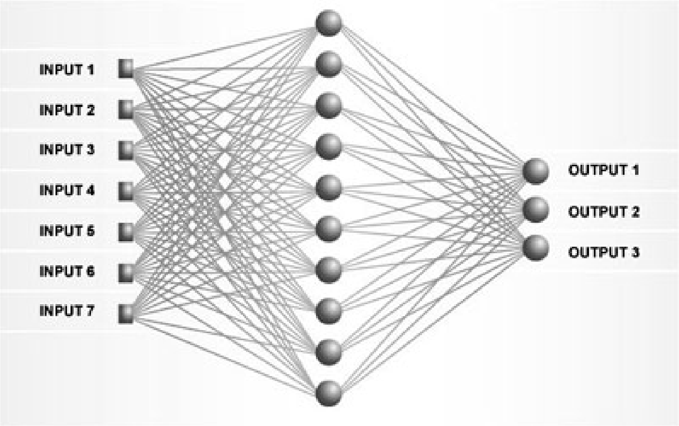


- The human brain has $\sim 10^{11}$ neurons, each connected to $\sim 10^{4}$ others.
- Inputs come from the dendrites, are aggregated in the soma, if the neuron starts firing impulses propagated to other neurons via axons.

- Neurons learn by changing the connection strengths of their synapses.
- Information storage is the nervous system is "distributed".
- The response time of the brain is quite fast, so the "depth" of the network can't be very great. (clear layer by layer organization in the visual system).

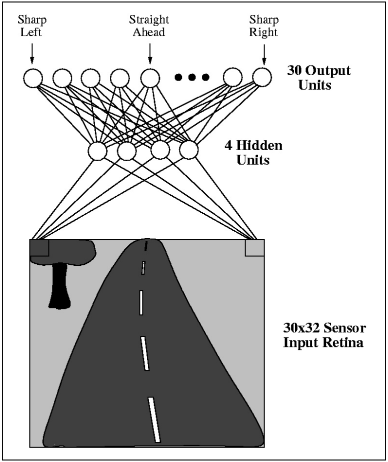
IDEA: Humans seem to be okay at learning, so why not try to replicate this in a computer? Goes back to the early days of AI, many successes and failures.

Multilayer artificial neural net



Question: But what should the individual neurons do and how should they learn?

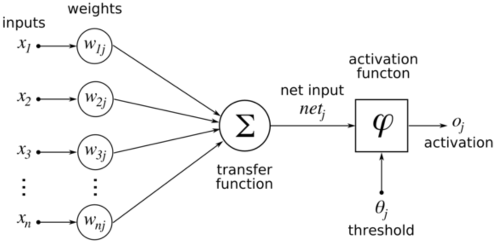
A success: ALVINN [Pomerleau ’95]



Drove unassisted from Pittsburgh to NYC on the highway.

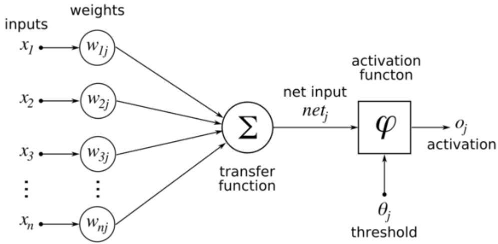
A model neuron

$$o = \sigma\Big(\theta + \sum_p w_p x_p\Big),$$



A model neuron

$$o = \sigma\Big(\theta + \sum_p w_p x_p\Big),$$

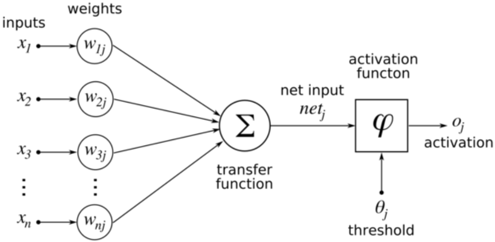


Notation:

- x_i : the i'th input

A model neuron

$$o = \sigma\left(\theta + \sum_p w_p x_p\right),$$

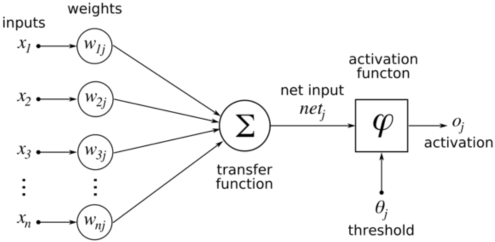


Notation:

- x_i : the i'th input
- w_i : the corresponding synaptic weight

A model neuron

$$o = \sigma\Big(\theta + \sum_p w_p x_p\Big),$$

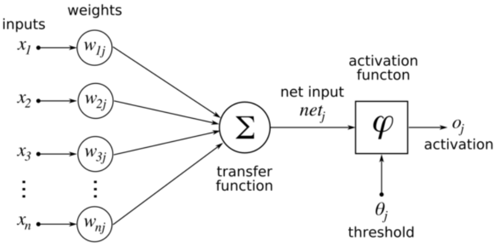


Notation:

- x_i : the i'th input
- w_i : the corresponding synaptic weight
- θ : the bias (can be eliminated as in the perceptron)

A model neuron

$$o = \sigma\big(\theta + \sum_p w_p x_p\big),$$



Notation:

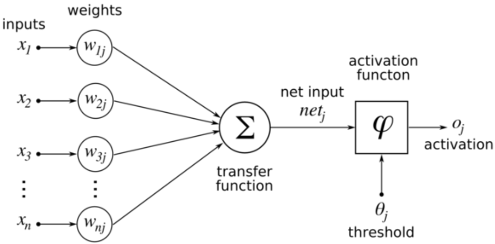
- x_i : the i'th input
- w_i : the corresponding synaptic weight
- θ : the bias (can be eliminated as in the perceptron)
- o : the output

Learning rule: Define a loss function $\ell(\mathbf{y}, \widehat{\mathbf{y}})$ for the entire network, and change the weights so as to reduce it. For example, for regression one can use

$$\ell(y, \widehat{y}) = (y - \widehat{y})^2.$$

A model neuron

$$o = \sigma\Big(\theta + \sum_p w_p x_p\Big),$$



Notation:

- x_i : the i'th input
- w_i : the corresponding synaptic weight
- θ : the bias (can be eliminated as in the perceptron)
- o : the output

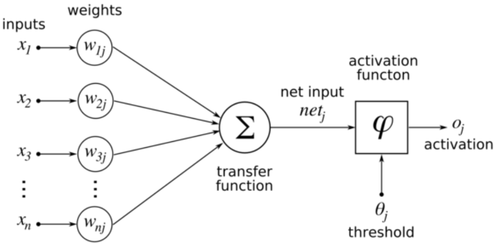
Learning rule: Define a loss function $\ell(\mathbf{y}, \widehat{\mathbf{y}})$ for the entire network, and change the weights so as to reduce it. For example, for regression one can use

$$\ell(y, \widehat{y}) = (y - \widehat{y})^2.$$

The behavior of the network is determined by

A model neuron

$$o = \sigma\big(\theta + \sum_p w_p x_p\big),$$



Notation:

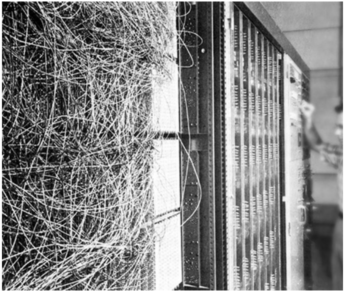
- x_i : the i'th input
- w_i : the corresponding synaptic weight
- θ : the bias (can be eliminated as in the perceptron)
- o : the output

Learning rule: Define a loss function $\ell(\mathbf{y}, \widehat{\mathbf{y}})$ for the entire network, and change the weights so as to reduce it. For example, for regression one can use

$$\ell(y, \widehat{y}) = (y - \widehat{y})^2.$$

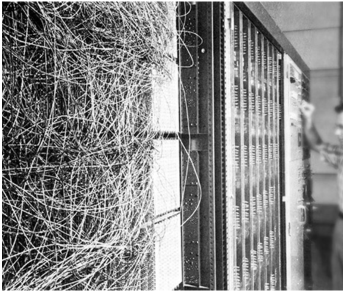
The behavior of the network is determined by

Multilayer Perceptron



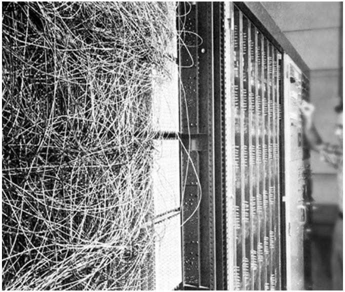
- Multilayer perceptron was thought of as universal model of the brain

Multilayer Perceptron



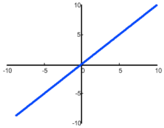
- Multilayer perceptron was thought of as universal model of the brain
- Mark I: 400 pixel image, 2020 photocells

Multilayer Perceptron



- Multilayer perceptron was thought of as universal model of the brain
- Mark I: 400 pixel image, 2020 photocells
- Problem: hard to train, multilayer perceptron plagued with local minima

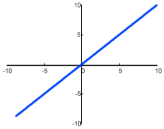
Activation functions



Linear: $\sigma(t) = t$

Question: What is the problem with this?

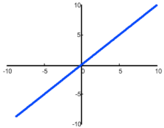
Activation functions



Linear: $\sigma(t) = t$

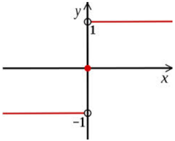
Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.

Activation functions



Linear: $\sigma(t) = t$

Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.



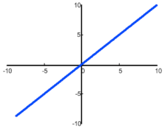
Hard threshold: $\sigma(t) = \mathrm{sgn}(t)$

"Threshold Logic Unit" [McCulloch & Pitts, 1943]

$\rightarrow$ Perceptron [Rosenblatt, 1958]

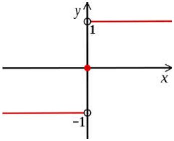
Question: What is the problem with this?

Activation functions



Linear: $\sigma(t) = t$

Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.



Hard threshold: $\sigma(t) = \operatorname{sgn}(t)$

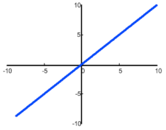
"Threshold Logic Unit" [McCulloch & Pitts, 1943]

$\rightarrow$ Perceptron [Rosenblatt, 1958]

Question: What is the problem with this?

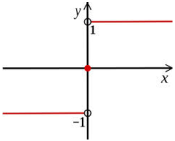
Not differentiable.

Activation functions



Linear: $\sigma(t) = t$

Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.



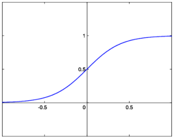
Hard threshold: $\sigma(t) = \mathrm{sgn}(t)$

"Threshold Logic Unit" [McCulloch & Pitts, 1943]

$\rightarrow$ Perceptron [Rosenblatt, 1958]

Question: What is the problem with this?

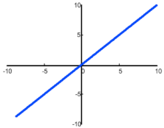
Not differentiable.



(log-)sigmoid: $\sigma(t) = 1/(1 + e^{-t})$

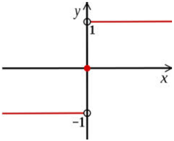
Also called the logistic function.

Activation functions



Linear: $\sigma(t) = t$

Question: What is the problem with this? Linear functions composed with each other are still linear, so no point in having a multilayer network.



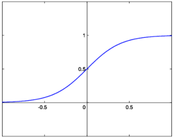
Hard threshold: $\sigma(t) = \mathrm{sgn}(t)$

"Threshold Logic Unit" [McCulloch & Pitts, 1943]

$\rightarrow$ Perceptron [Rosenblatt, 1958]

Question: What is the problem with this?

Not differentiable.

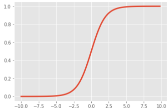


(log-)sigmoid: $\sigma(t) = 1/(1 + e^{-t})$

Also called the logistic function.

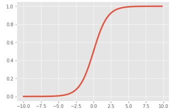
This is what we will use.

Activation functions

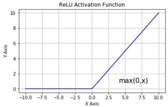


tanh: $\sigma(x) = \tanh(x) = (e^{x} - e^{-x})/(e^{x} + e^{-x})$

Activation functions

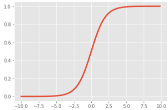


tanh: $\sigma(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$

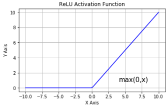


Rectified linear unit (ReLU): $\sigma(x) = \max(0, x)$

Activation functions

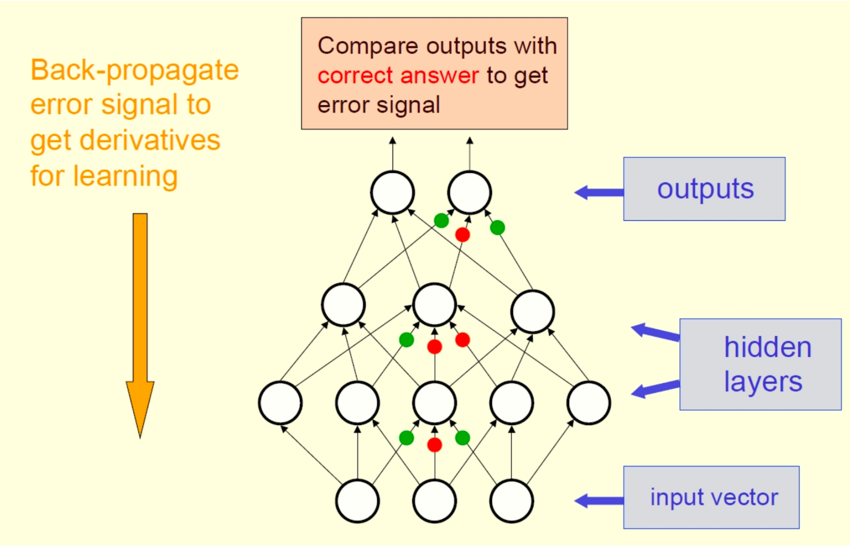


tanh: $\sigma(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$



Rectified linear unit (ReLU): $\sigma(x) = \max(0, x)$

Multilayer Neural Nets



Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]

Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]

Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]
- In principle, the more layers the better.

Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]
- In principle, the more layers the better. But training deep nets is hard $\rightarrow$ went out of fashion 1990-2006.

Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]
- In principle, the more layers the better. But training deep nets is hard $\rightarrow$ went out of fashion 1990-2006.
- Reasons for resurgence of deep learning:
  - Efficient greedy pretraining idea [Hinton et al., 2006]

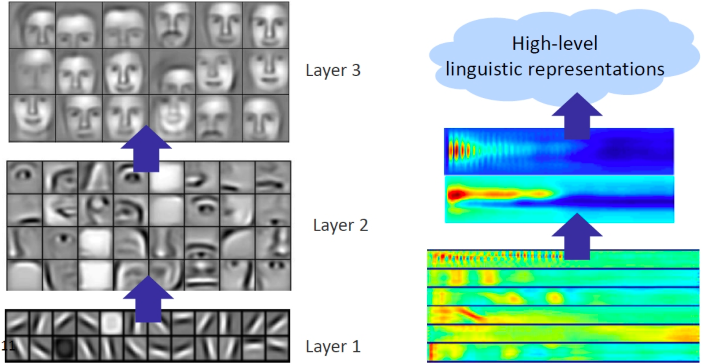
Multilayer Neural Nets

- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]
- In principle, the more layers the better. But training deep nets is hard $\rightarrow$ went out of fashion 1990-2006.
- Reasons for resurgence of deep learning:
  - Efficient greedy pretraining idea [Hinton et al., 2006]
  - Explosion in amount of training data.

Multilayer Neural Nets

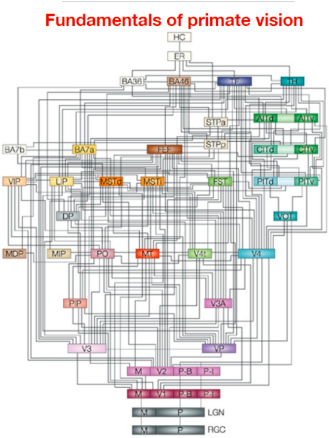
- Theoretical result: 2-layer net with linear output can approximate any continuous function over compact domain to arbitrary accuracy (given enough hidden units!) [Cybenko 1989]
- But: there are functions with poly-size logic gate circuit of depth k that require exponential size when restricted to depth $O(k)$ [Hastad, 1986]
- In principle, the more layers the better. But training deep nets is hard $\rightarrow$ went out of fashion 1990-2006.
- Reasons for resurgence of deep learning:
  - Efficient greedy pretraining idea [Hinton et al., 2006]
  - Explosion in amount of training data.
  - Modern GPU architectures make training much faster.

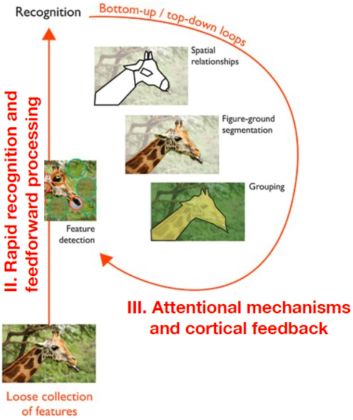
Multilayer Representations



For vision tasks in particular, representing complex scenes in terms of a hierarchy of features makes sense.

Multilayer Representations





Feed-forward NN for regression

For regression $f : \mathbb{R}^n \to \mathbb{R}^m$:

- n neurons in input layer (layer 0)

Feed-forward NN for regression

For regression $f : \mathbb{R}^n \to \mathbb{R}^m$:

- n neurons in input layer (layer 0)
- m neurons in output layer (layer L)

Feed-forward NN for regression

For regression $f: \mathbb{R}^n \to \mathbb{R}^m$:

- n neurons in input layer (layer 0)
- m neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers

Feed-forward NN for regression

For regression $f : \mathbb{R}^n \to \mathbb{R}^m$:

- n neurons in input layer (layer 0)
- m neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers
- Use a loss function like

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{d} (a_{\tau(i)} - y_i)^2,$$

where $a_{\tau(i)}$ is the output of the i 'th neuron in the output layer.

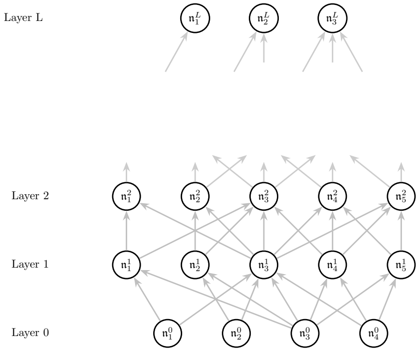
Feed-forward NN for regression

For regression $f : \mathbb{R}^n \to \mathbb{R}^m$:

- n neurons in input layer (layer 0)
- m neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers
- Use a loss function like

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{d} (a_{\tau(i)} - y_i)^2,$$

  where $a_{\tau(i)}$ is the output of the i'th neuron in the output layer.
- Each neuron may be connected to all the neurons in the previous layer or just a subset of them.



Feed-forward NN for classification

For regression $f\colon \mathbb{R}^n \to \{1, 2, \ldots, k\}$:

- n neurons in input layer (layer 0)

Feed-forward NN for classification

For regression $f : \mathbb{R}^n \to \{1, 2, \ldots, k\}$:

- n neurons in input layer (layer 0)
- k neurons in output layer (layer L)

Feed-forward NN for classification

For regression $f\colon \mathbb{R}^n \to \{1, 2, \ldots, k\}$:

- n neurons in input layer (layer 0)
- k neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers

Feed-forward NN for classification

For regression $f : \mathbb{R}^n \to \{1, 2, \ldots, k\}$:

- n neurons in input layer (layer 0)
- k neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers
- Use a loss function like the log-softmax

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = -\log\left(\frac{e^{a_{\tau(i)}}}{\sum_{i=1}^{k} e^{a_{\tau(i)}}}\right),$$

where $a_{\tau(i)}$ is the output of the i'th neuron in the output layer.

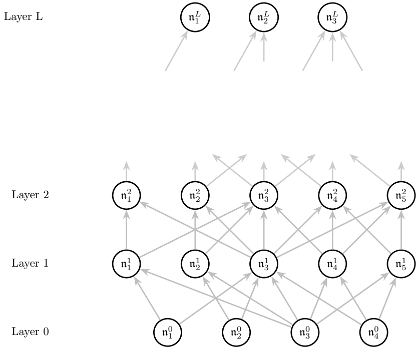
Feed-forward NN for classification

For regression $f: \mathbb{R}^n \to \{1, 2, \ldots, k\}$:

- n neurons in input layer (layer 0)
- k neurons in output layer (layer L)
- Some number of neurons of your choice in the intermediate (hidden) layers
- Use a loss function like the log-softmax

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = -\log\left(\frac{e^{a_{\tau(i)}}}{\sum_{i=1}^{k} e^{a_{\tau(i)}}}\right),$$

  where $a_{\tau(i)}$ is the output of the i'th neuron in the output layer.
- Each neuron may be connected to all the neurons in the previous layer or just a subset of them.



Training NNs with backpropagation

Training NNs

- Present training examples one by one (online learning).

Training NNs

- Present training examples one by one (online learning).
- The error on an example $(\mathbf{x}, \mathbf{y})$ is some function of the difference between the desired and actual output of the last layer, e.g., for a multivariate regression task

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{d} (a_{\tau(i)} - y_i)^2,$$

where $\tau(i)$ is the index of the output neuron that is supposed to predict y_i.

Training NNs

- Present training examples one by one (online learning).
- The error on an example $(\mathbf{x}, \mathbf{y})$ is some function of the difference between the desired and actual output of the last layer, e.g., for a multivariate regression task

$$\mathcal{E}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \sum_{i=1}^{d} (a_{\tau(i)} - y_i)^2,$$

  where $\tau(i)$ is the index of the output neuron that is supposed to predict y_i.
- Adjust each weight of each neuron in each layer by gradient descent

$$w_{s \to t} \leftarrow w_{s \to t} - \eta \, \frac{\partial \mathcal{E}}{\partial w_{s \to t}},$$

  where η is a parameter called the **learning rate**.

Notation

Consider a feed-forward architecture with L layers:

- Set of neurons in layer ℓ: $\mathcal{N}_\ell$

Notation

Consider a feed-forward architecture with L layers:

- Set of neurons in layer ℓ: $\mathcal{N}_\ell$
- Weight of connection from neuron s to neuron t: $w_{s\to t}$
- Output (**activation**) of any neuron t in layer ℓ:

$$a_t = \sigma\underbrace{\Bigg(\sum_{s\in\mathcal{I}(t)} w_{s\to t}\, a_s + b_t\Bigg)}_{z_t} = \sigma(z_t),$$

where $\mathcal{I}_t \subseteq \mathcal{N}_{\ell-1}$ is the set of neurons feeding into t (in a fully connected feed-foward network $\mathcal{I}_t = \mathcal{N}_{\ell-1}$).

Notation

Consider a feed-forward architecture with L layers:

- Set of neurons in layer ℓ: $\mathcal{N}_\ell$
- Weight of connection from neuron s to neuron t: $w_{s\to t}$
- Output (**activation**) of any neuron t in layer ℓ:

$$a_t = \sigma\underbrace{\Big(\sum_{s\in\mathcal{I}(t)} w_{s\to t}\, a_s + b_t\Big)}_{z_t} = \sigma(z_t),$$

  where $\mathcal{I}_t \subseteq \mathcal{N}_{\ell-1}$ is the set of neurons feeding into t (in a fully connected feed-foward network $\mathcal{I}_t = \mathcal{N}_{\ell-1}$).
- Similarly, $\mathcal{O}_t$ is the set of neurons in the next layer that t feeds into.

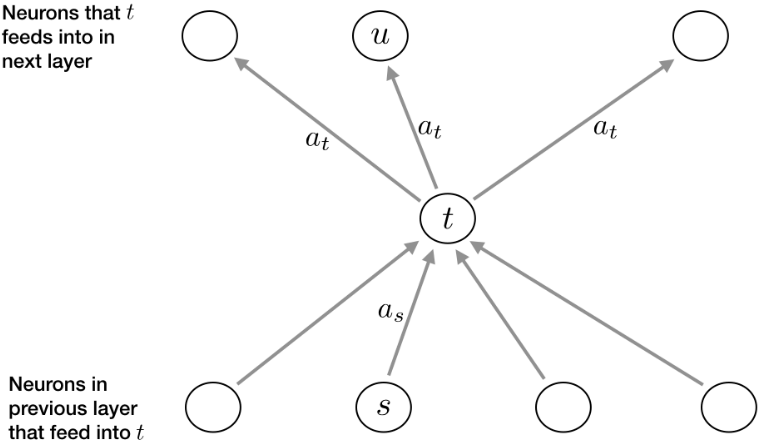
Notation

Consider a feed-forward architecture with L layers:

- Set of neurons in layer ℓ: $\mathcal{N}_\ell$
- Weight of connection from neuron s to neuron t: $w_{s\to t}$
- Output (**activation**) of any neuron t in layer ℓ:

$$a_t = \sigma\underbrace{\Big(\sum_{s\in\mathcal{I}(t)} w_{s\to t}\, a_s + b_t\Big)}_{z_t} = \sigma(z_t),$$

  where $\mathcal{I}_t \subseteq \mathcal{N}_{\ell-1}$ is the set of neurons feeding into t (in a fully connected feed-foward network $\mathcal{I}_t = \mathcal{N}_{\ell-1}$).
- Similarly, $\mathcal{O}_t$ is the set of neurons in the next layer that t feeds into.
- The "pre-activation" z_t plays an important role in the following.


Neurons that t feeds into in next layer
u
a_t
a_t
a_t
t
a_s
Neurons in previous layer that feed into t
s

Backpropagation

Problem: how to compute the $\frac{\partial \mathcal{E}}{\partial w_{s \to t}}$ derivatives efficiently for all the weights in the neural network?

- Define the **error** (slight misnomer)

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}.$$

Backpropagation

Problem: how to compute the $\frac{\partial \mathcal{E}}{\partial w_{s\to t}}$ derivatives efficiently for all the weights in the neural network?

- Define the **error** (slight misnomer)

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}.$$

- Using the chain rule, we then have the very simple relations

$$\frac{\partial \mathcal{E}}{\partial w_{s\to t}} = \frac{\partial \mathcal{E}}{\partial z_t}\frac{\partial z_t}{\partial w_{s\to t}} = \delta_t \cdot a_s, \qquad \frac{\partial \mathcal{E}}{\partial b_t} = \delta_t.$$

Backpropagation

Problem: how to compute the $\frac{\partial \mathcal{E}}{\partial w_{s \to t}}$ derivatives efficiently for all the weights in the neural network?

- Define the **error** (slight misnomer)

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}.$$

- Using the chain rule, we then have the very simple relations

$$\frac{\partial \mathcal{E}}{\partial w_{s \to t}} = \frac{\partial \mathcal{E}}{\partial z_t} \frac{\partial z_t}{\partial w_{s \to t}} = \delta_t \cdot a_s, \qquad \frac{\partial \mathcal{E}}{\partial b_t} = \delta_t.$$

So now the question becomes: how do we compute δ_t for all neurons in the network simultaneously?

Backpropagation

Assume that δ_u errrors have been computed for for all neurons in layer $\ell + 1$. Then for any neuron t in layer ℓ,

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t} = \sum_{u \in \mathcal{O}_t} \underbrace{\frac{\partial \mathcal{E}}{\partial z_u}}_{\delta_t} \frac{\partial z_u}{\partial z_t}.$$

Backpropagation

Assume that δ_u errrors have been computed for for all neurons in layer $\ell + 1$. Then for any neuron t in layer ℓ,

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t} = \sum_{u \in \mathcal{O}_t} \underbrace{\frac{\partial \mathcal{E}}{\partial z_u}}_{\delta_t} \frac{\partial z_u}{\partial z_t}.$$

and

$$\frac{\partial z_u}{\partial z_t} = \frac{\partial z_u}{\partial a_t} \frac{\partial a_t}{\partial z_t} = w_{t \to u}\, \sigma'(z_t).$$

Backpropagation

Assume that δ_u errrors have been computed for for all neurons in layer $\ell+1$. Then for any neuron t in layer ℓ,

$$\delta_t = \frac{\partial \mathcal{E}}{\partial z_t} = \sum_{u \in \mathcal{O}_t} \underbrace{\frac{\partial \mathcal{E}}{\partial z_u}}_{\delta_t} \frac{\partial z_u}{\partial z_t}.$$

and

$$\frac{\partial z_u}{\partial z_t} = \frac{\partial z_u}{\partial a_t} \frac{\partial a_t}{\partial z_t} = w_{t \to u}\, \sigma'(z_t).$$

So we have the simple backpropagation formula

$$\delta_t = \sum_{u \in \mathcal{O}_t} w_{t \to u}\, \sigma'(z_t)\, \delta_u.$$

In the last layer, δ_t is computed directly from the loss.

Forward pass

- Initialize the activations of the neurons in layer 0 based on the input $\mathbf{x}$.

Forward pass

- Initialize the activations of the neurons in layer 0 based on the input $\mathbf{x}$.
- For layers $1, 2, \ldots L$ compute the activations from the activations of the previous layer by

$$a_t = \sigma\Big(\sum_{s \in \mathcal{I}(t)} w_{s \to t}\, a_s + b_t\Big) = \sigma(z_t).$$

Forward pass

- Initialize the activations of the neurons in layer 0 based on the input $\mathbf{x}$.
- For layers $1, 2, \ldots L$ compute the activations from the activations of the previous layer by

$$a_t = \sigma\Big(\sum_{s \in \mathcal{I}(t)} w_{s \to t}\, a_s + b_t\Big) = \sigma(z_t).$$

- Read off $\widehat{\mathbf{y}}$ from the last layer, and compute the loss $\mathcal{E}(\widehat{\mathbf{y}}, \mathbf{y})$.

Backward pass

- Initialize the errors in the last layer by directly computing $\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}$.

Backward pass

- Initialize the errors in the last layer by directly computing $\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}$.
- For layers $L-1, L-2, \ldots, 1$ compute the errors from layer above by

$$\delta_t = \sum_{u \in \mathcal{O}_t} w_{t \to u}\, \sigma'(z_t)\, \delta_u.$$

Backward pass

- Initialize the errors in the last layer by directly computing $\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}$.
- For layers $L-1, L-2, \ldots, 1$ compute the errors from layer above by

$$\delta_t = \sum_{u \in \mathcal{O}_t} w_{t \to u} \, \sigma'(z_t) \, \delta_u.$$

- For each neuron in each layer compute the partial derivatives that we ultimately need

$$\frac{\partial \mathcal{E}}{\partial w_{s \to t}} = \delta_t \cdot a_s, \qquad \frac{\partial \mathcal{E}}{\partial b_t} = \delta_t.$$

Backward pass

- Initialize the errors in the last layer by directly computing $\delta_t = \frac{\partial \mathcal{E}}{\partial z_t}$.
- For layers $L-1, L-2, \dots, 1$ compute the errors from layer above by

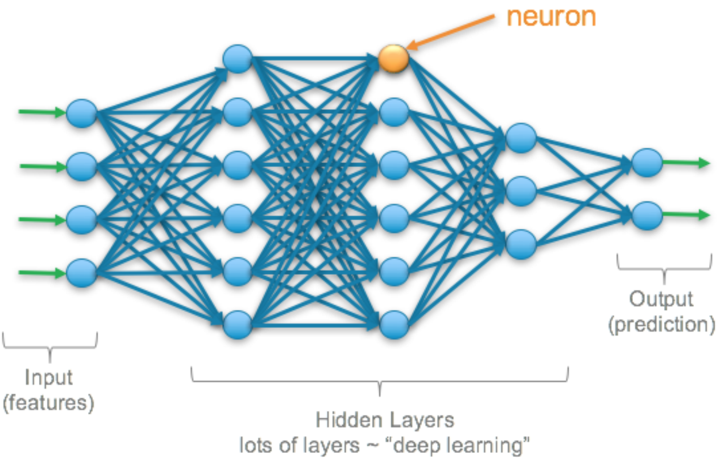
$$\delta_t = \sum_{u \in \mathcal{O}_t} w_{t \to u}\, \sigma'(z_t)\, \delta_u.$$

- For each neuron in each layer compute the partial derivatives that we ultimately need

$$\frac{\partial \mathcal{E}}{\partial w_{s \to t}} = \delta_t \cdot a_s, \qquad \frac{\partial \mathcal{E}}{\partial b_t} = \delta_t.$$

- Update the weights and biases by the SGD rule.

The multi-layer perceptron



A feed-forward network where is layer is fully connected is called a multi-layer perceptron.

Forward pass

For a MLP, the forward iteration can be expressed as

$$\boldsymbol{a}_\ell = \sigma(\underbrace{W_\ell\, \boldsymbol{a}_{\ell-1}}_{\boldsymbol{z}_\ell} + \boldsymbol{b}_\ell),$$

where

- $\boldsymbol{a}_\ell$ is the vector activations in layer ℓ,
- $\boldsymbol{b}_\ell$ is the vector of biases in layer ℓ,
- W_ℓ is the matrix of weights from layer $\ell - 1$ to layer ℓ.

Forward pass

For a MLP, the forward iteration can be expressed as

$$\boldsymbol{a}_\ell = \sigma(\underbrace{W_\ell\, \boldsymbol{a}_{\ell-1}}_{\boldsymbol{z}_\ell} + \boldsymbol{b}_\ell),$$

where

- $\boldsymbol{a}_\ell$ is the vector activations in layer ℓ,
- $\boldsymbol{b}_\ell$ is the vector of biases in layer ℓ,
- W_ℓ is the matrix of weights from layer $\ell - 1$ to layer ℓ.

The forward pass consists of just computing $\mathbf{x} \mapsto \boldsymbol{a}_1 \mapsto \boldsymbol{a}_2 \mapsto \ldots \mapsto \boldsymbol{a}_L \mapsto \widehat{y} \mapsto \mathcal{E}$.

Backward pass

Setting $\boldsymbol{\delta}_\ell$ as the vector of errors in layer ℓ, the backward iteration is

$$\boldsymbol{\delta}_\ell = \sigma'(\boldsymbol{z}_t) \odot (W_\ell^\top \boldsymbol{\delta}_{\ell+1}),$$

where $\odot$ is the elementwise product of two vectors.

Backward pass

Setting $\boldsymbol{\delta}_\ell$ as the vector of errors in layer ℓ, the backward iteration is

$$\boldsymbol{\delta}_\ell = \sigma'(\boldsymbol{z}_t) \odot (W_\ell^\top \boldsymbol{\delta}_{\ell+1}),$$

where $\odot$ is the elementwise product of two vectors.

The backward pass consists of computing $\mathcal{E} \mapsto \boldsymbol{\delta}_L \mapsto \boldsymbol{\delta}_{L-1} \mapsto \ldots \mapsto \boldsymbol{\delta}_1$.

Backward pass

Setting $\boldsymbol{\delta}_\ell$ as the vector of errors in layer ℓ, the backward iteration is

$$\boldsymbol{\delta}_\ell = \sigma'(\boldsymbol{z}_t) \odot (W_\ell^\top \boldsymbol{\delta}_{\ell+1}),$$

where $\odot$ is the elementwise product of two vectors.

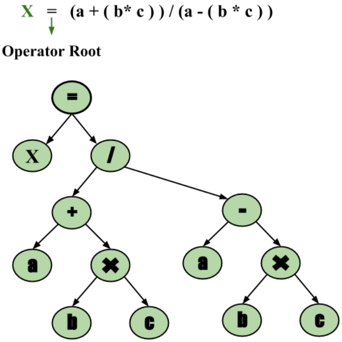
The backward pass consists of computing $\mathcal{E} \mapsto \boldsymbol{\delta}_L \mapsto \boldsymbol{\delta}_{L-1} \mapsto \ldots \mapsto \boldsymbol{\delta}_1$.

The individual $\partial \mathcal{E} / \partial w_{s \to t}$ and $\partial \mathcal{E} / \partial b_t$ derivatives are computed as before.

Differentiable computing

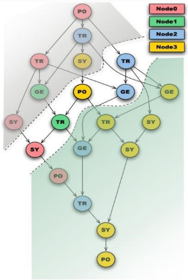
Computation graphs

It is an old idea in compilers to use a directed acyclic graph (DAG) as an intermediate representation (IR).



Similarly, in high performance computing (HPC) large DAGs are used to depict the interdependencies between parts of a massive compute job.

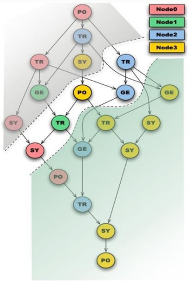
Computation graphs



The DAG can be used to

- Optimize the order in which computations are performed.

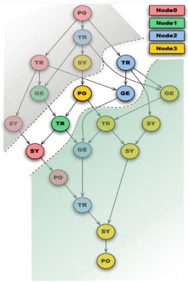
Computation graphs



The DAG can be used to

- Optimize the order in which computations are performed.
- Optimize the way in which parts of the computation are allocated to processors/nodes.

Computation graphs



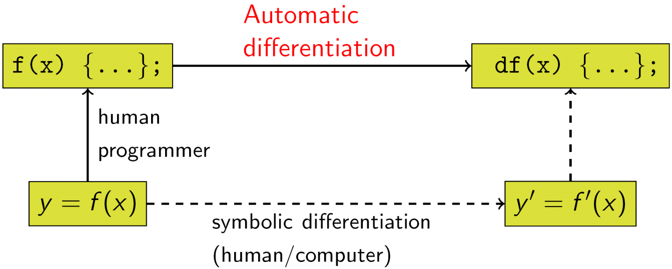
The DAG can be used to

- Optimize the order in which computations are performed.
- Optimize the way in which parts of the computation are allocated to processors/nodes.

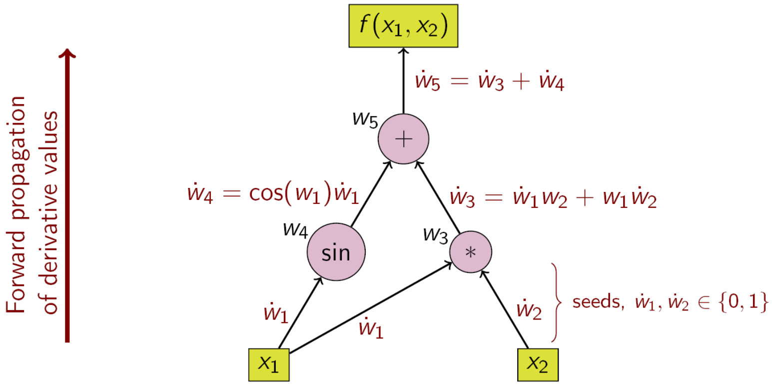
All this can be done statically (ahead of time) or dynamically (at runtime).

Automatic differentation

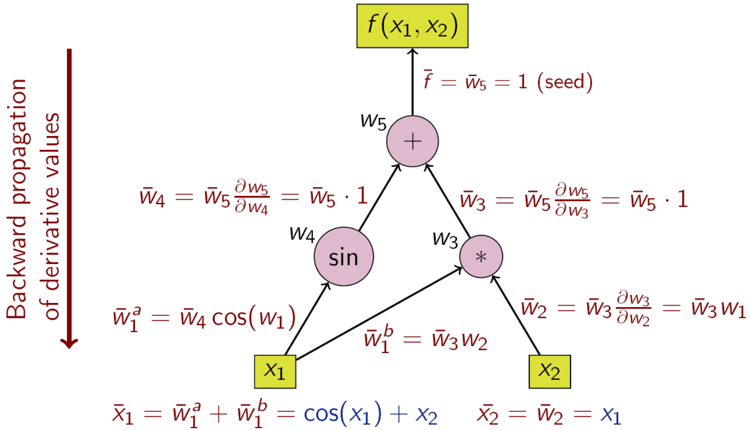
Another interesting field deals with writing compilers that can compute the derivative of any user defined (differentiable) function.



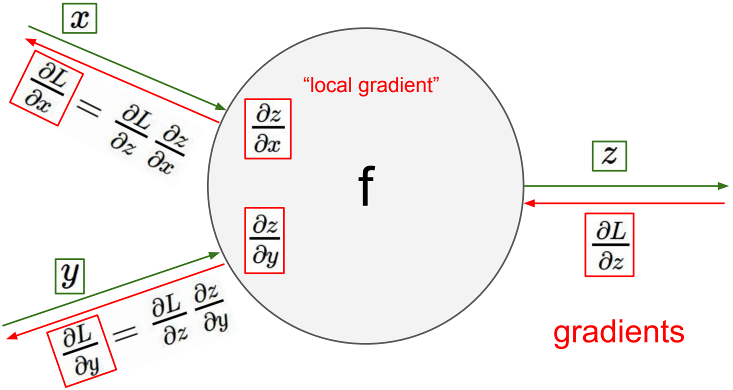
Automatic differentiation



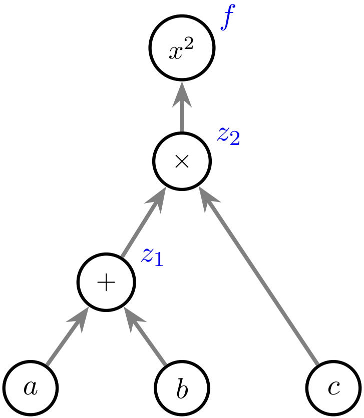
Automatic differentiation



Modern deep learning frameworks combine this idea with DAG-based runtimes.

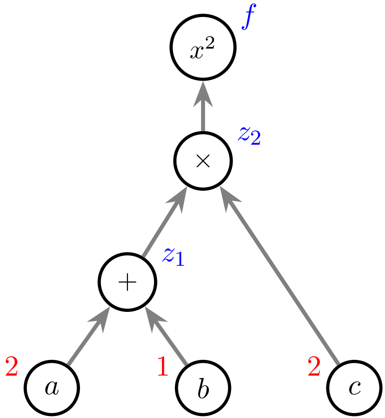

x
y
z
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
$\frac{\partial L}{\partial z}$
gradients
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$

Forward and backward computations

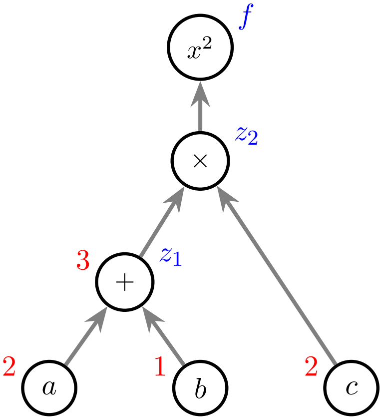


$$f(a, b, c) = ((a + b)c)^2$$

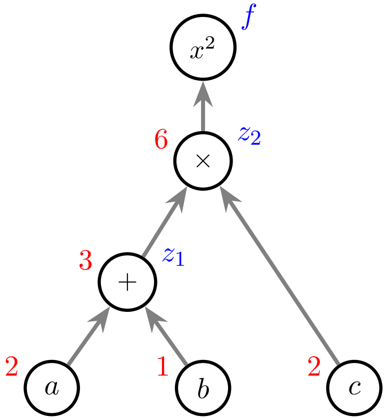
Forward



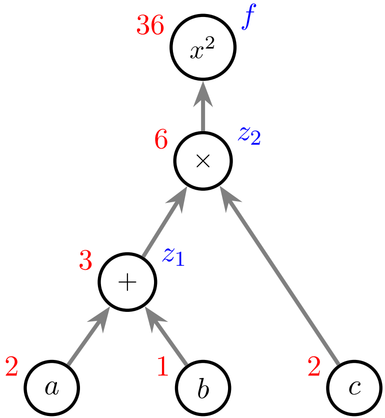
Forward



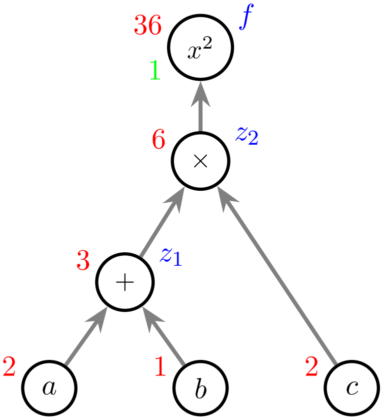
Forward



Forward

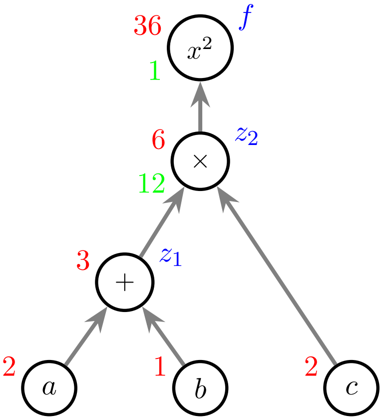


Backward



$$\frac{\partial f}{\partial f} = 1$$

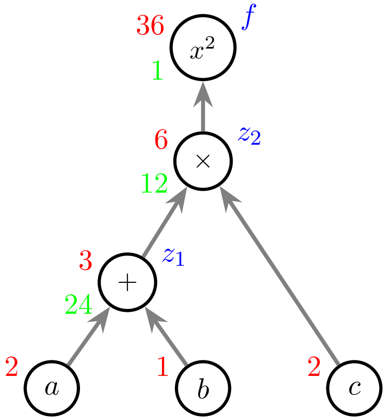
Backward



$$f = z_2^2$$

$$\frac{\partial f}{\partial z_2} = \underbrace{\frac{\partial f}{\partial f}}_{1} \underbrace{\frac{\partial f}{\partial z_2}}_{2z_2=12} = 12$$

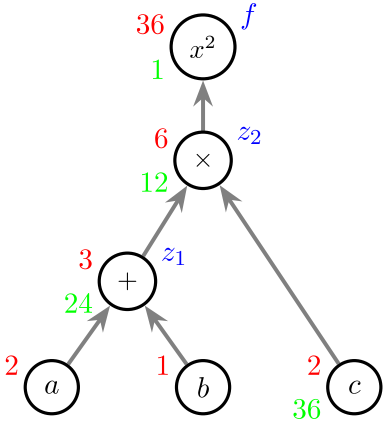
Backward



$$z_2 = z_1 \cdot c$$

$$\frac{\partial f}{\partial z_1} = \underbrace{\frac{\partial f}{\partial z_2}}_{12} \underbrace{\frac{\partial z_2}{\partial z_1}}_{c} = 24$$

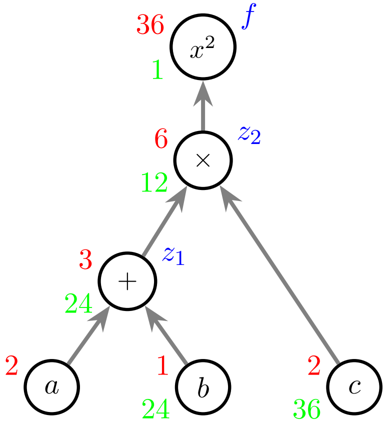
Backward



$$z_2 = z_1 \cdot c$$

$$\frac{\partial f}{\partial c} = \underbrace{\frac{\partial f}{\partial z_2}}_{12} \underbrace{\frac{\partial z_2}{\partial c}}_{z_1} = 36$$

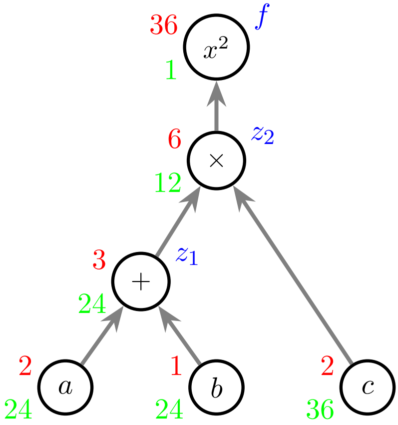
Backward



$$z_1 = a + b$$

$$\frac{\partial f}{\partial b} = \underbrace{\frac{\partial f}{\partial z_1}}_{24} \underbrace{\frac{\partial z_1}{\partial b}}_{1} = 24$$

Backward

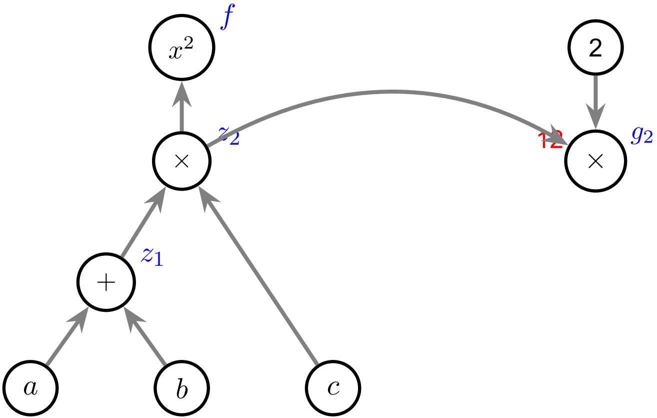


$$z_1 = a + b$$

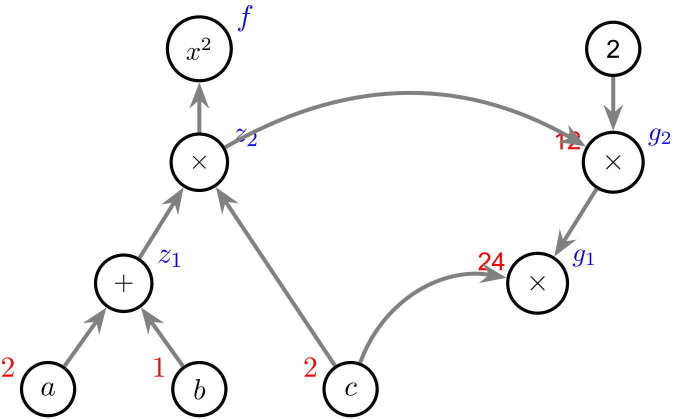
$$\frac{\partial f}{\partial a} = \underbrace{\frac{\partial f}{\partial z_1}}_{24} \underbrace{\frac{\partial z_1}{\partial a}}_{1} = 24$$

Symbolic differentiation

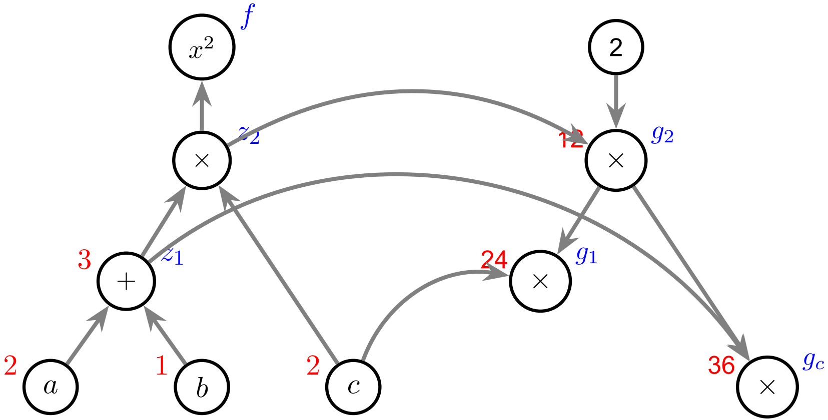
$$g_2 = \frac{\partial f}{\partial z_2} = 2 \cdot z_2$$



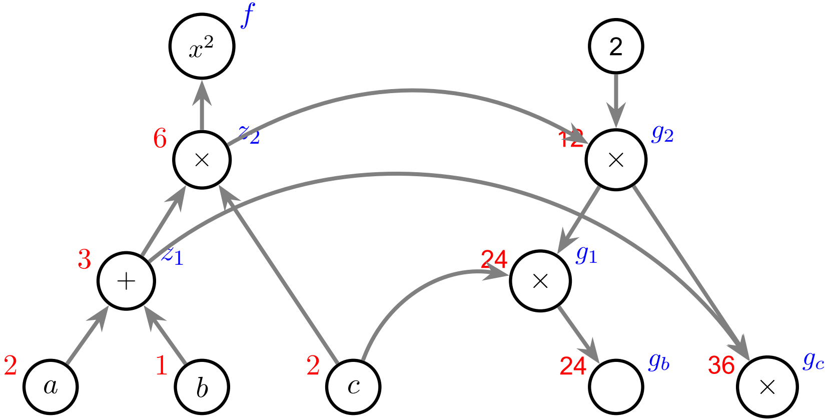
$$g_1 = \frac{\partial f}{\partial z_1} = \frac{\partial f}{\partial z_2} \underbrace{\frac{\partial z_2}{\partial z_1}}_{c}$$



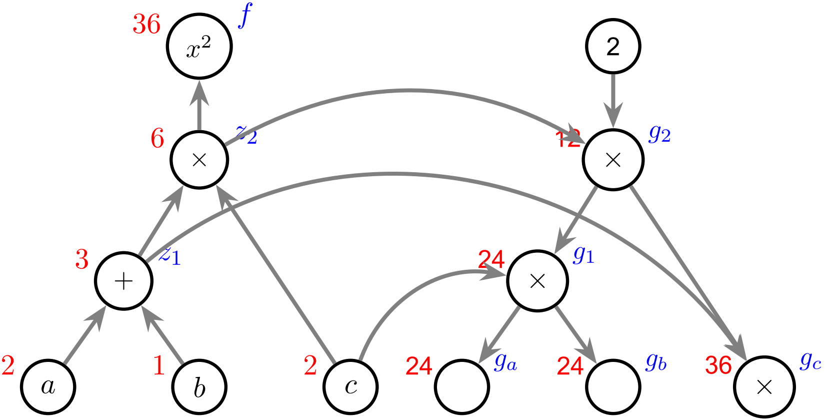
$$g_c = \frac{\partial f}{\partial c} = \frac{\partial f}{\partial z_2} \underbrace{\frac{\partial z_2}{\partial c}}_{z_1}$$



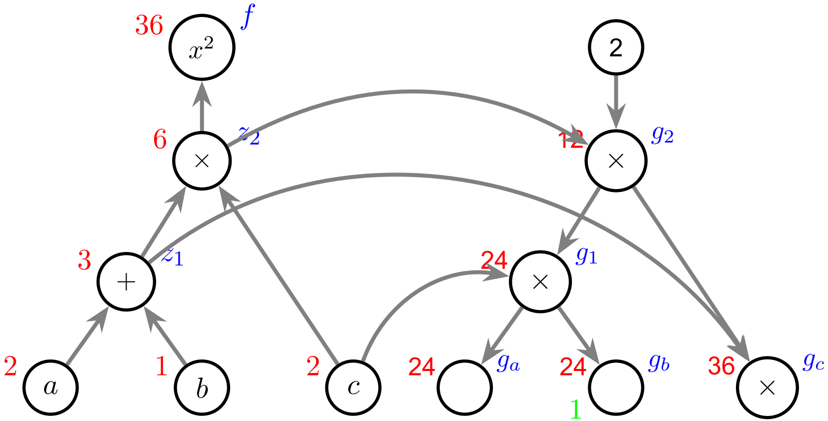
$$g_b = \frac{\partial f}{\partial b} = \frac{\partial f}{\partial z_1} \underbrace{\frac{\partial z_1}{\partial b}}_{1}$$



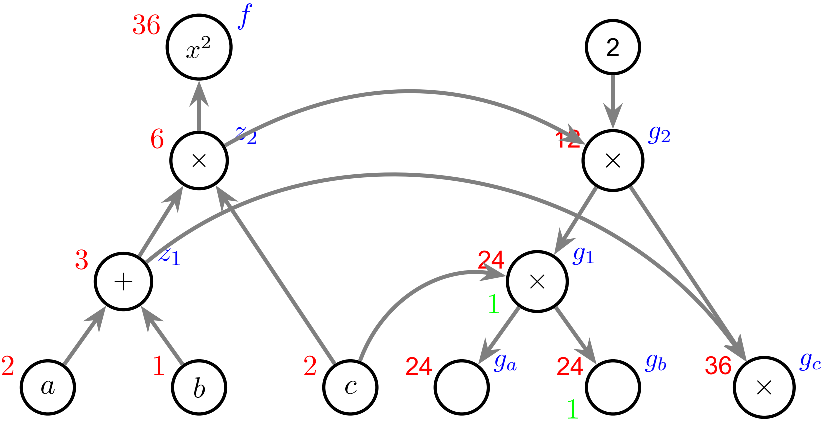
$$g_a = \frac{\partial f}{\partial a} = \frac{\partial f}{\partial z_1} \underbrace{\frac{\partial z_1}{\partial a}}_{1}$$



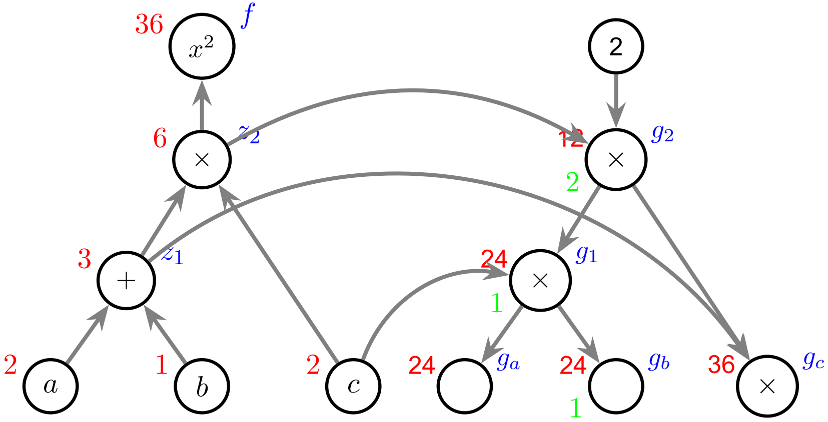
$$\frac{\partial g_b}{\partial g_b} = 1$$



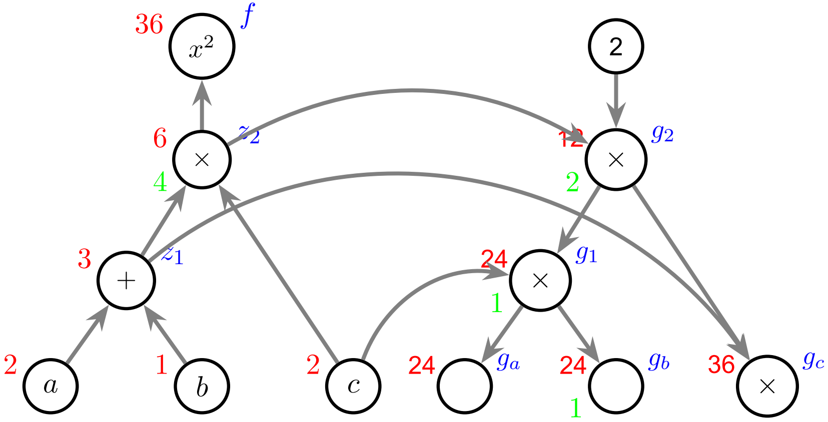
$$\frac{\partial g_b}{\partial g_1} = \frac{\partial g_b}{\partial g_b} = 1$$



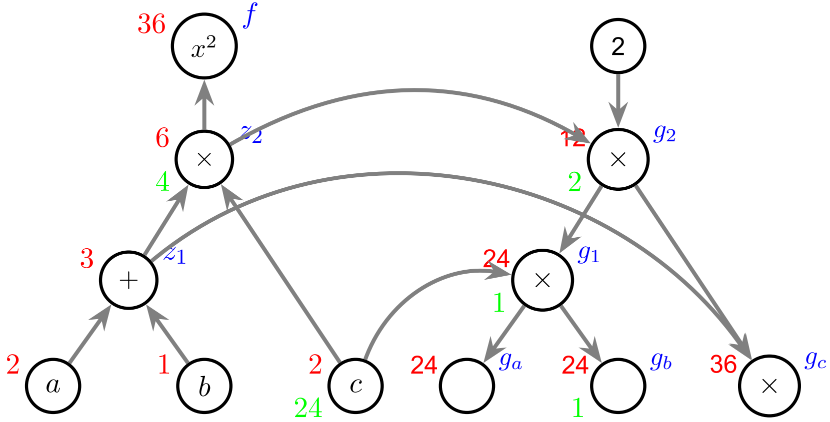
$$\frac{\partial g_b}{\partial g_2} = \underbrace{\frac{\partial g_b}{\partial g_1}}_{1} \underbrace{\frac{\partial g_1}{\partial g_2}}_{c} = 2$$



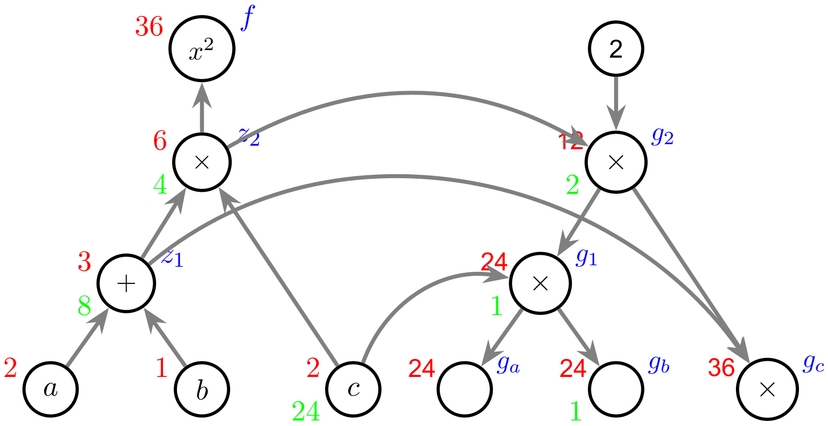
$$\frac{\partial g_b}{\partial z_2} = \underbrace{\frac{\partial g_b}{\partial g_2}}_{2} \underbrace{\frac{\partial g_2}{\partial z_2}}_{2} = 4$$



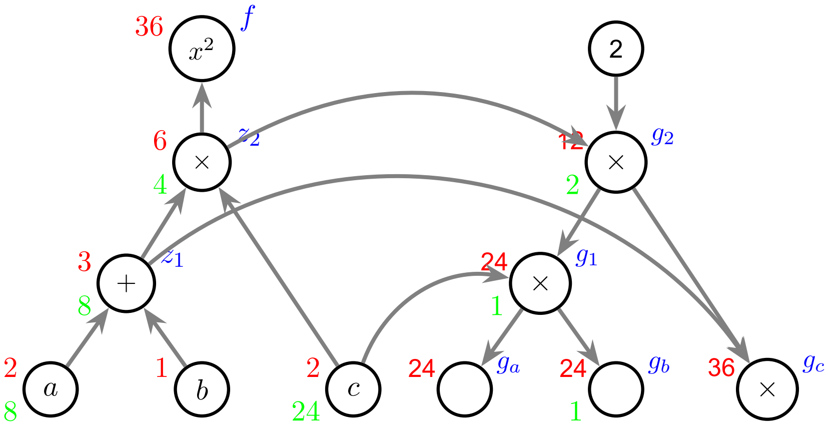
$$\frac{\partial g_b}{\partial c} = \underbrace{\frac{\partial g_b}{\partial z_2}}_{4}\underbrace{\frac{\partial z_2}{\partial c}}_{z_1} + \underbrace{\frac{\partial g_b}{\partial g_1}}_{1}\underbrace{\frac{\partial g_1}{\partial c}}_{g_2} = 4{\cdot}3+1{\cdot}12 = 24$$



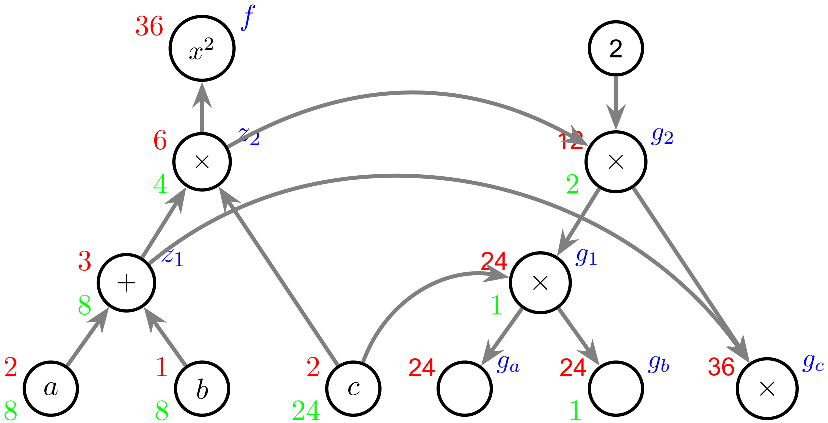
$$\frac{\partial g_b}{\partial z_1} = \underbrace{\frac{\partial g_b}{\partial z_2}}_{4} \underbrace{\frac{\partial z_2}{\partial z_1}}_{c} = 8$$



$$\frac{\partial g_b}{\partial a} = \underbrace{\frac{\partial g_b}{\partial z_1}}_{8} \underbrace{\frac{\partial z_1}{\partial a}}_{1} = 8$$



$$\frac{\partial g_b}{\partial b} = \underbrace{\frac{\partial g_b}{\partial z_1}}_{8} \underbrace{\frac{\partial z_1}{\partial b}}_{1} = 8$$



Verification

$$f = ((a+b)c)^2$$

$$\frac{\partial f}{\partial b} = 2(a+b)c^2$$

Verification

$$f = ((a+b)c)^2$$

$$\frac{\partial f}{\partial b} = 2(a+b)c^2$$

$$\frac{\partial f}{\partial a \partial b} = 2c^2 = 8$$

Verification

$$f = ((a+b)c)^2$$

$$\frac{\partial f}{\partial b} = 2(a+b)c^2$$

$$\frac{\partial f}{\partial a \partial b} = 2c^2 = 8$$

$$\frac{\partial f}{\partial a \partial c} = 2(a+b)c = 12$$