Neural Networks: Part II

Designing Neural Networks + PyTorch Intro

Daniel Yukimura

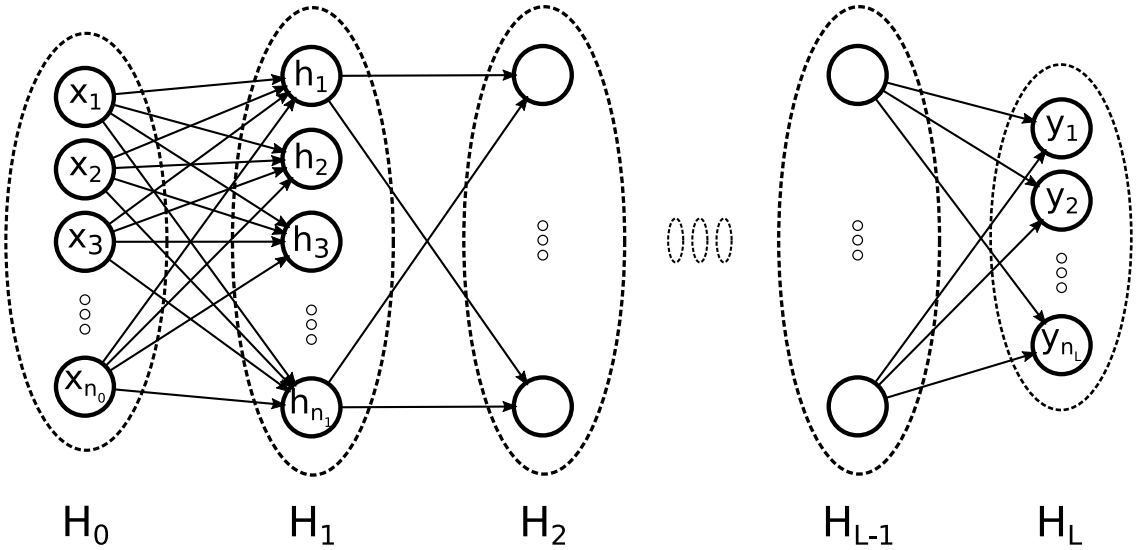
yukimura@impa.br

September 6, 2018

Neural Networks II

Recall last class:

Feedforward Neural Network (FNN)

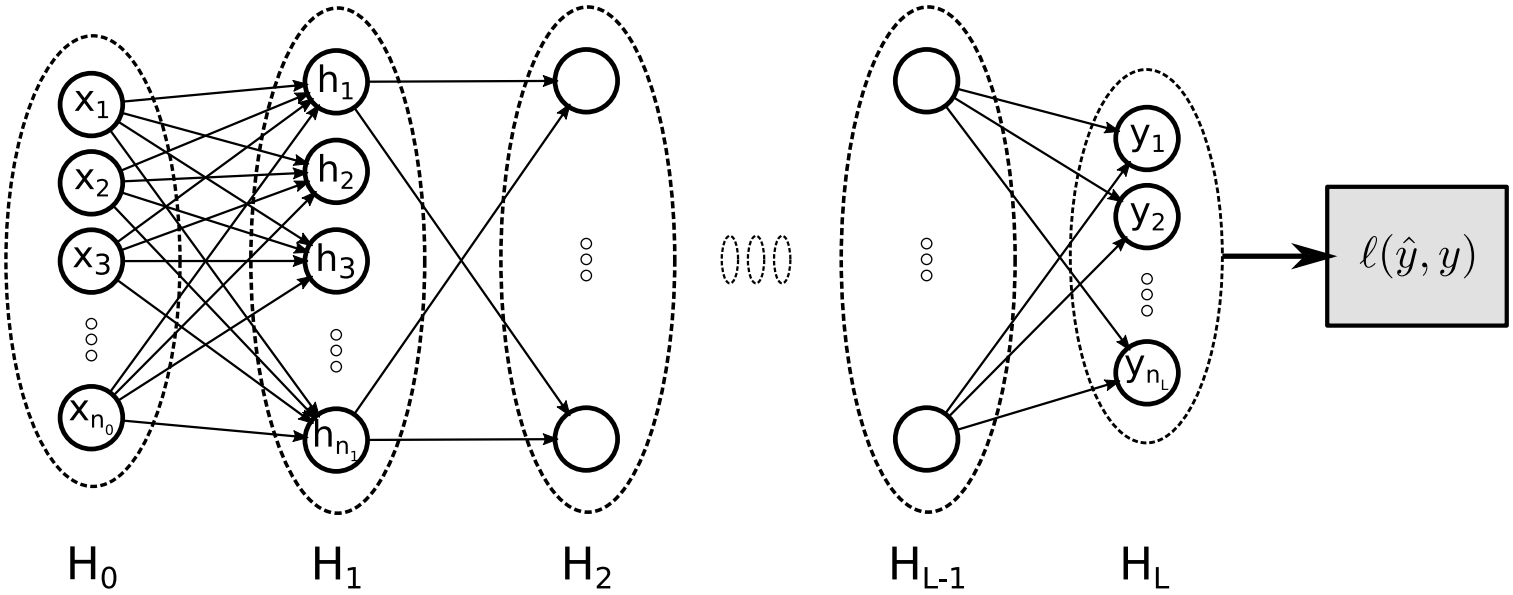


$$h^{(\ell)} = \sigma\left(W^{(\ell)^T} h^{(\ell-1)} + b^{(\ell)}\right)$$

Neural Networks II

Recall last class:

Empirical Risk Minimization



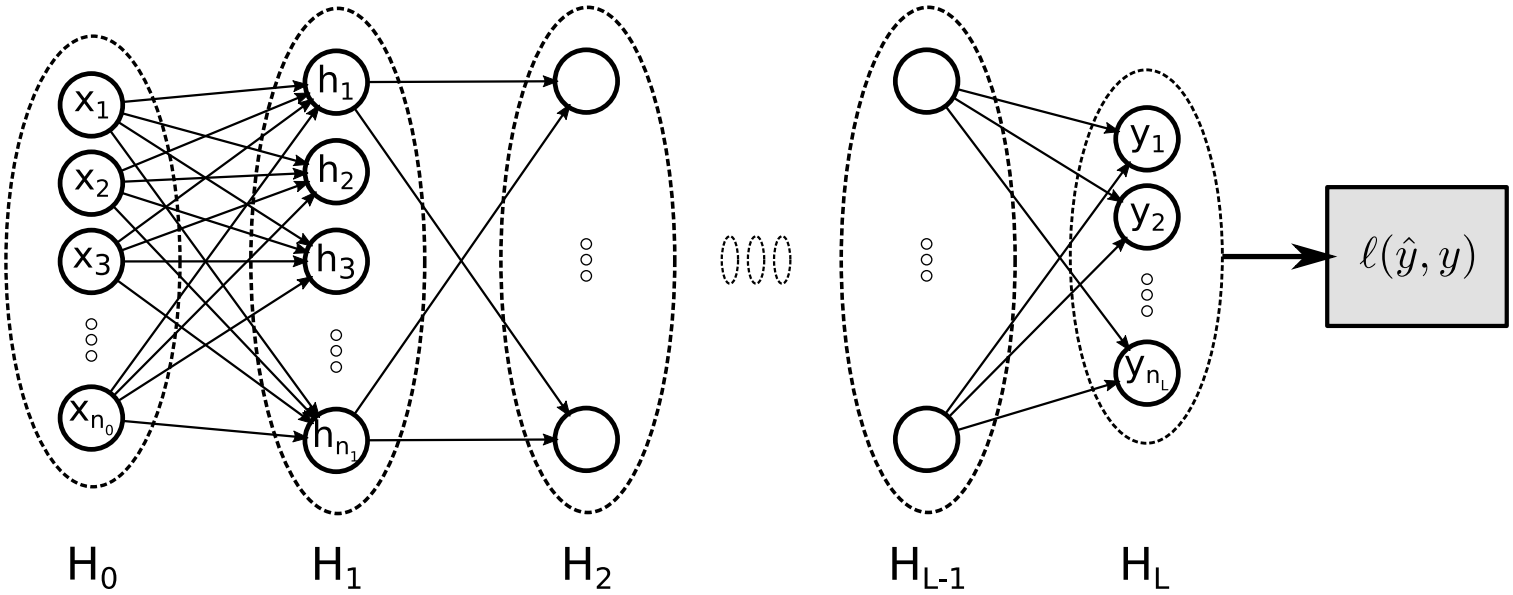
$$L_n(\theta) = \sum_{i=1}^{n} \ell\left(f(x_i, \theta), y_i\right)$$

$$\hat{\theta}_n = \operatorname*{argmin}_{\theta \in \Theta} L_n(\theta)$$

Neural Networks II

Recall last class:

Stochastic Gradient Descent

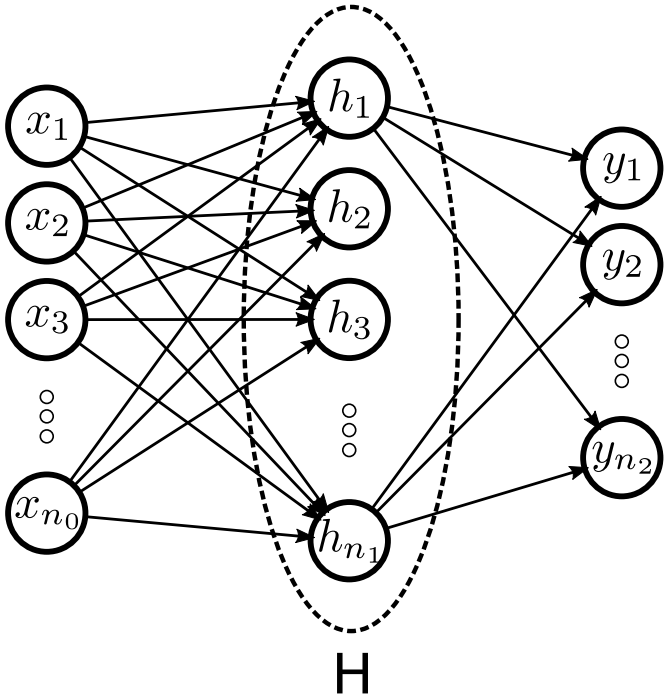


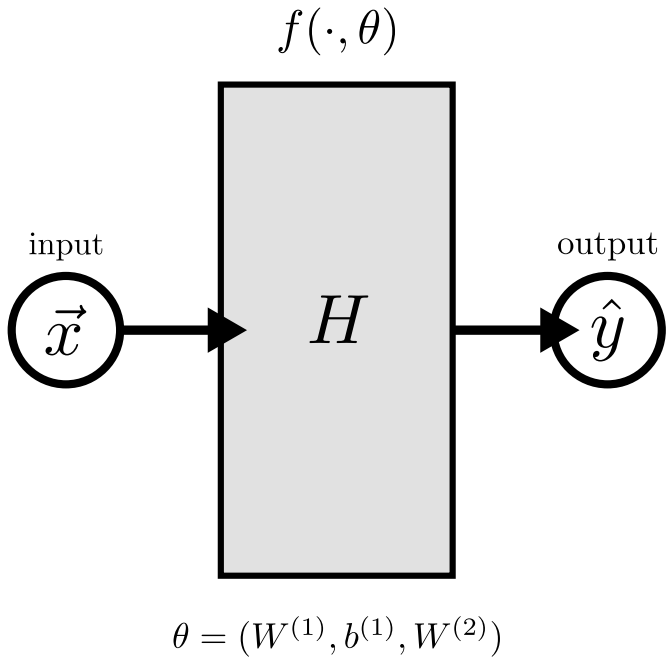
$$\hat{L}_S(\theta) = \frac{1}{S} \sum_{i \in \mathcal{S}} \ell(\theta, Z_i)$$

$$\theta_{t+1} = \theta_t - \alpha \nabla \hat{L}_S(\theta)$$

Designing Neural Networks

Modular approach:

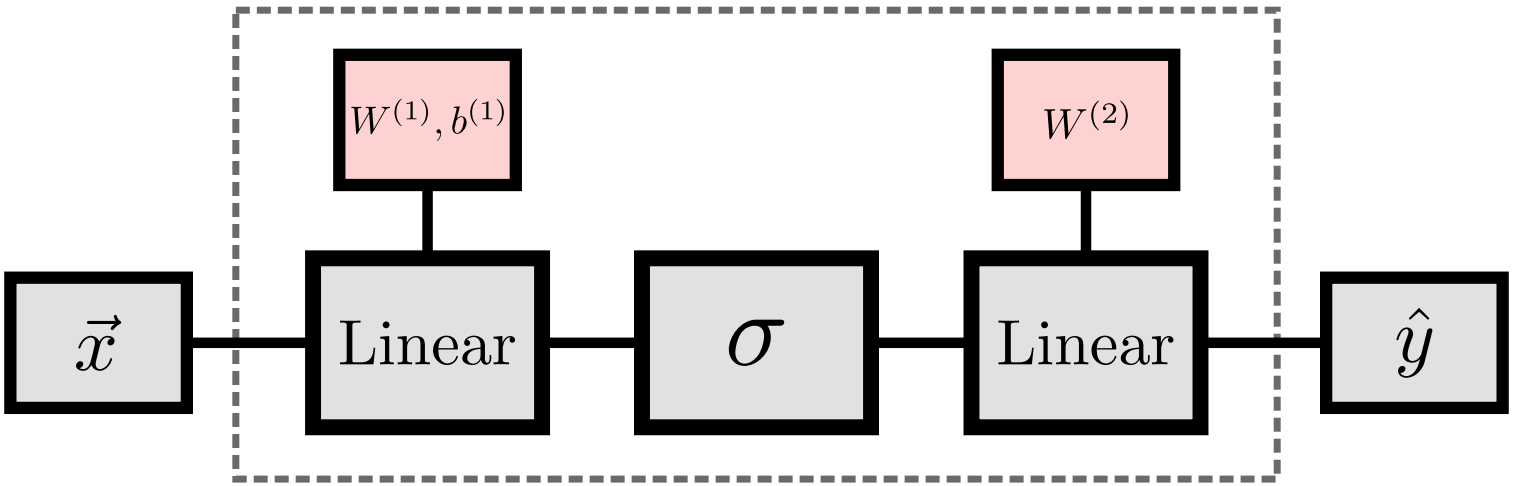




Designing Neural Networks

Modular approach:

Modularity allow us to design complex structures from simpler ones.

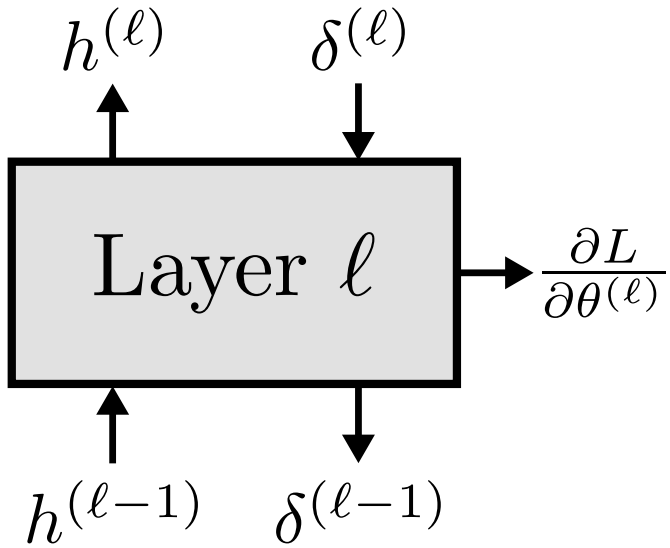


Designing Neural Networks

Backpropagation:

The steps of the backpropagation algorithm can also be encapsulated inside the modules.

- $h^{(\ell)} = f_\ell(h^{(\ell-1)}, \theta^{(\ell)})$
- $\frac{\partial L}{\partial \theta^{(\ell)}} = \frac{\partial L}{\partial h^{(\ell)}} \cdot \frac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}} = \delta^{(\ell)} \frac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}}$
- $\delta^{(\ell-1)} = \delta^{(\ell)} \cdot \frac{\partial h^{(\ell)}}{\partial h^{(\ell-1)}}$

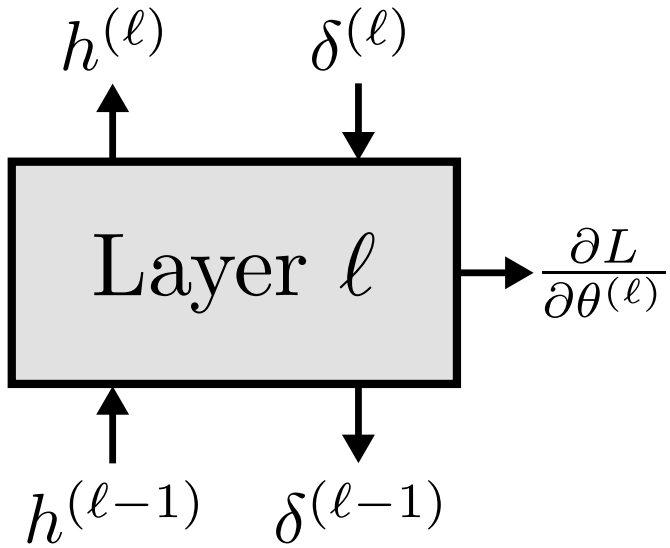


Designing Neural Networks

Backpropagation:

The steps of the backpropagation algorithm can also be encapsulated inside the modules.

- $h^{(\ell)} = f_\ell(h^{(\ell-1)}, \theta^{(\ell)})$
- $\dfrac{\partial L}{\partial \theta^{(\ell)}} = \dfrac{\partial L}{\partial h^{(\ell)}} \cdot \dfrac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}} = \delta^{(\ell)} \dfrac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}}$
- $\delta^{(\ell-1)} = \delta^{(\ell)} \cdot \dfrac{\partial h^{(\ell)}}{\partial h^{(\ell-1)}}$

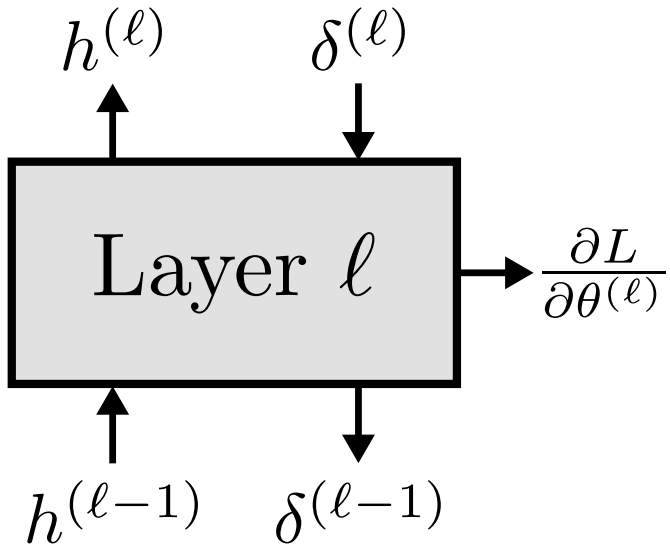


Designing Neural Networks

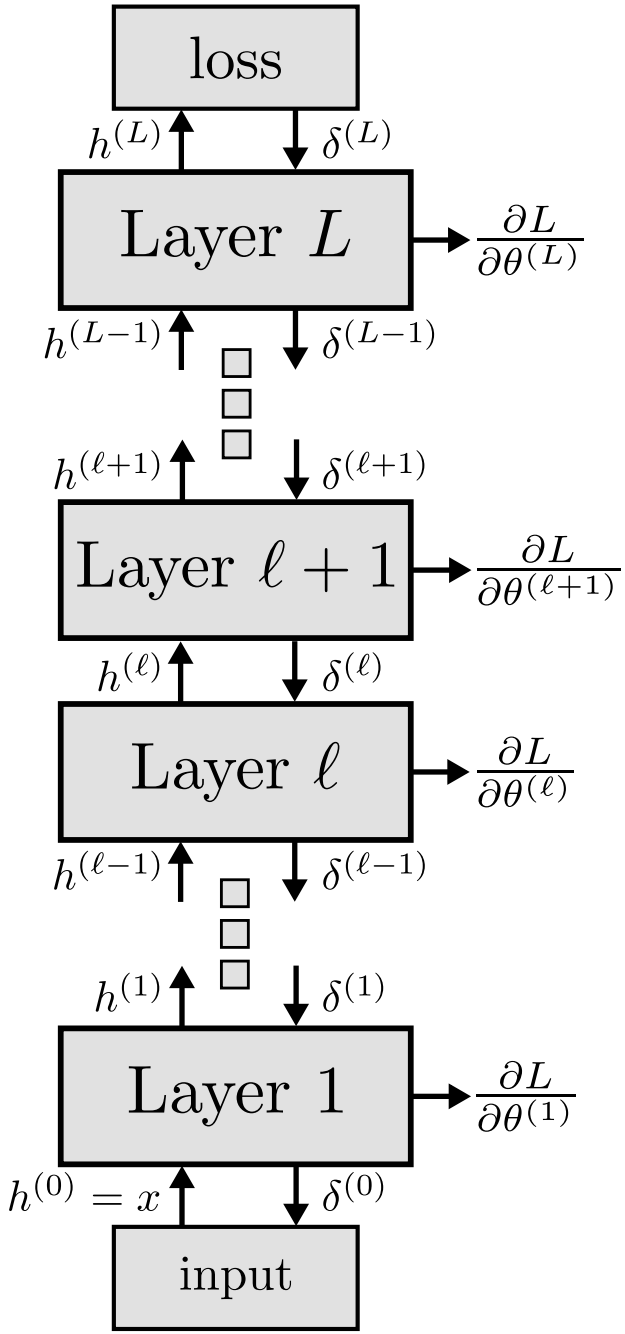
Backpropagation:

The steps of the backpropagation algorithm can also be encapsulated inside the modules.

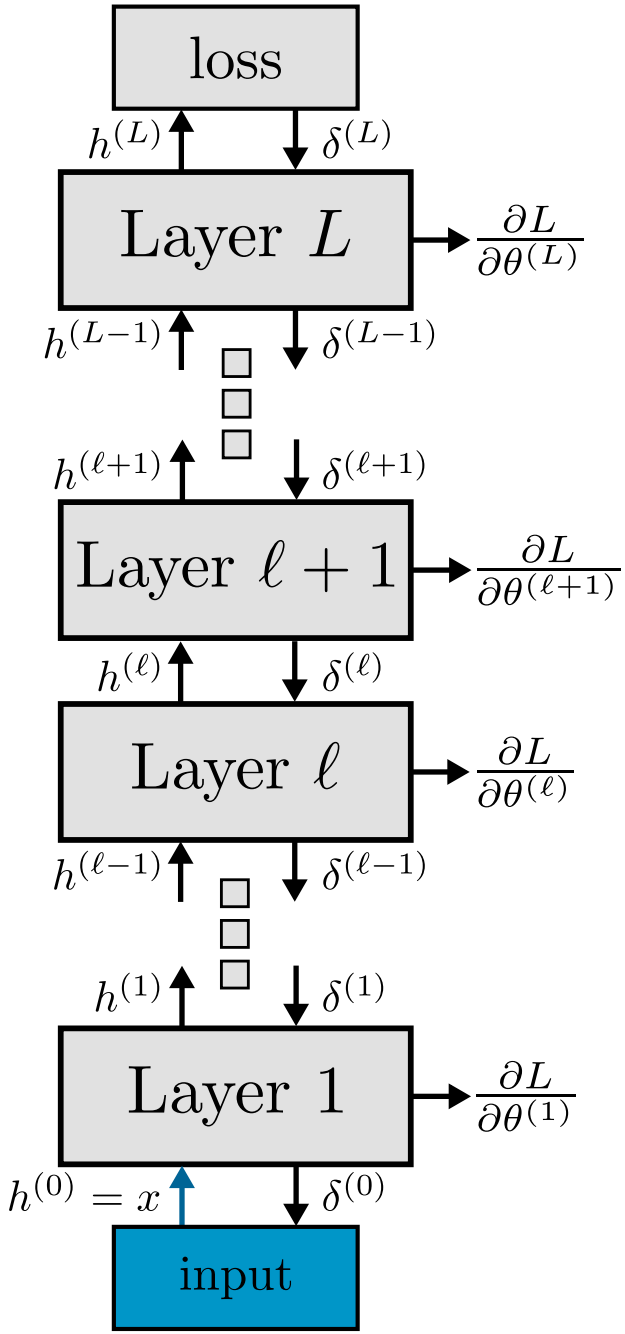
- $h^{(\ell)} = f_\ell(h^{(\ell-1)}, \theta^{(\ell)})$
- $\dfrac{\partial L}{\partial \theta^{(\ell)}} = \dfrac{\partial L}{\partial h^{(\ell)}} \cdot \dfrac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}} = \delta^{(\ell)} \dfrac{\partial h^{(\ell)}}{\partial \theta^{(\ell)}}$
- $\delta^{(\ell-1)} = \delta^{(\ell)} \cdot \dfrac{\partial h^{(\ell)}}{\partial h^{(\ell-1)}}$



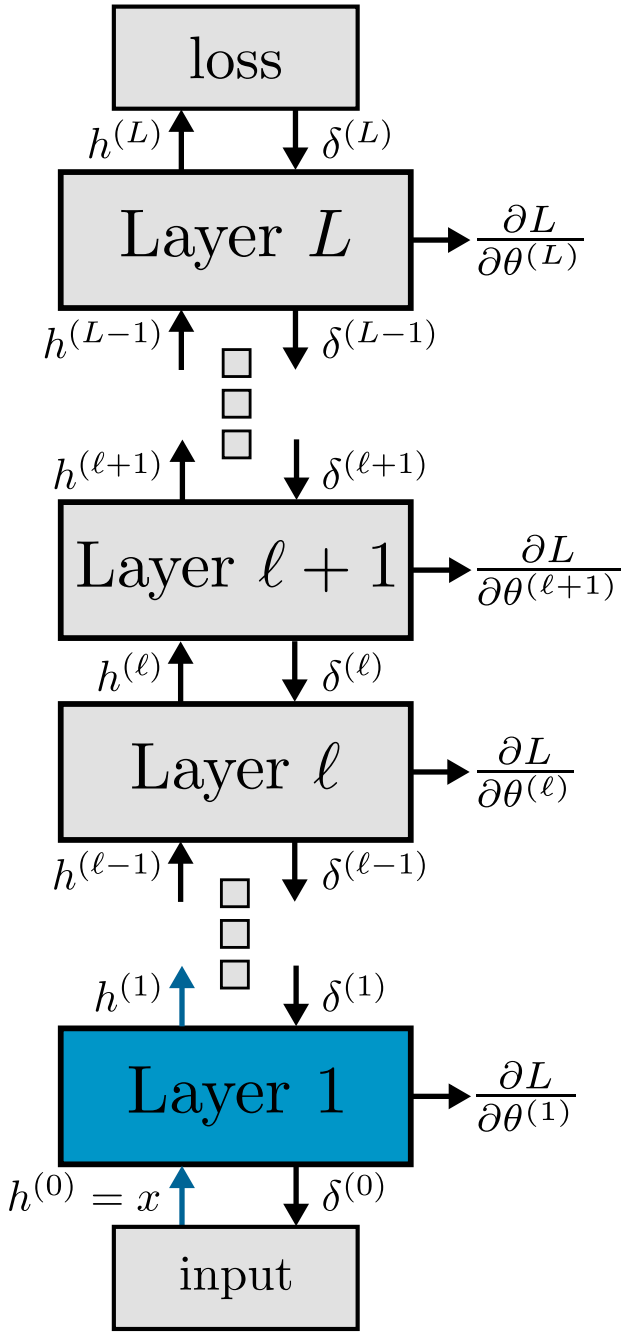
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and backpropagation are the upwards and downwards flows, respectively.



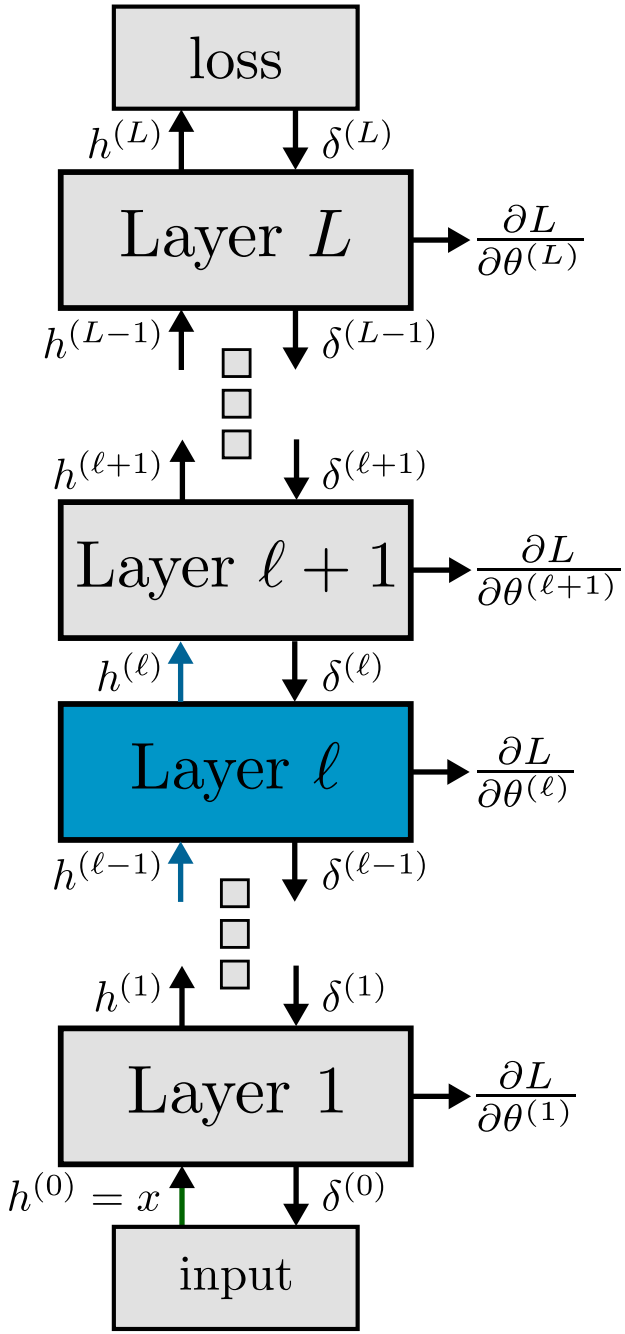
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



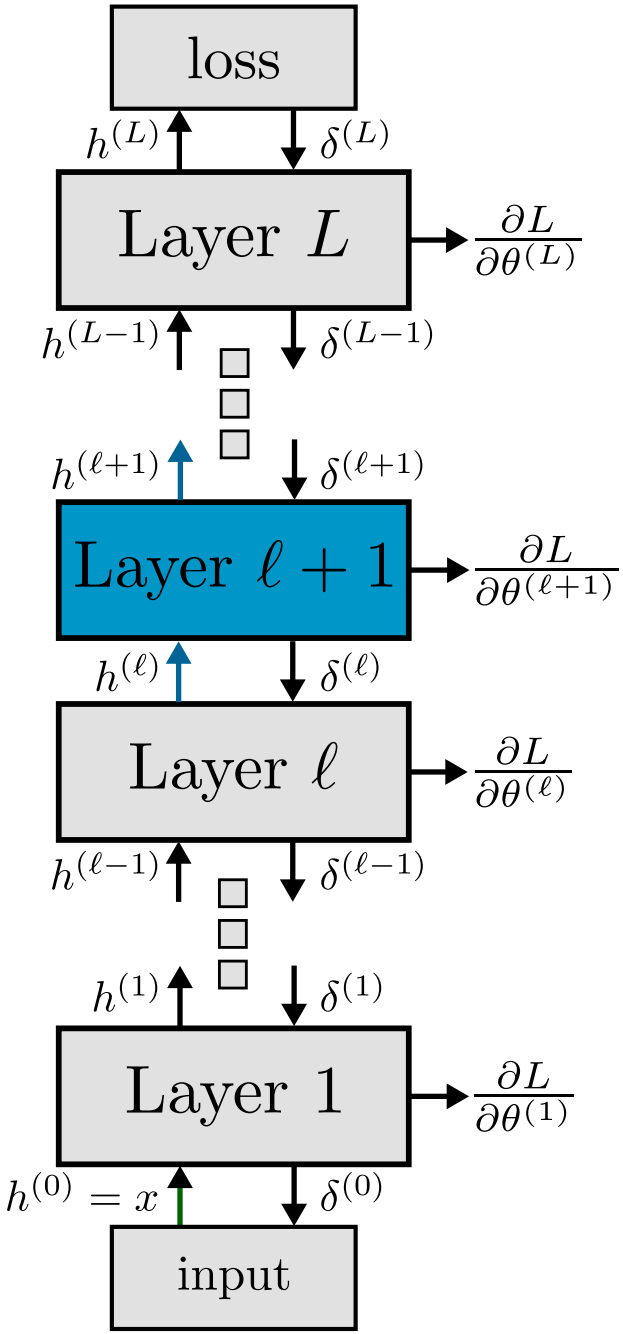
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



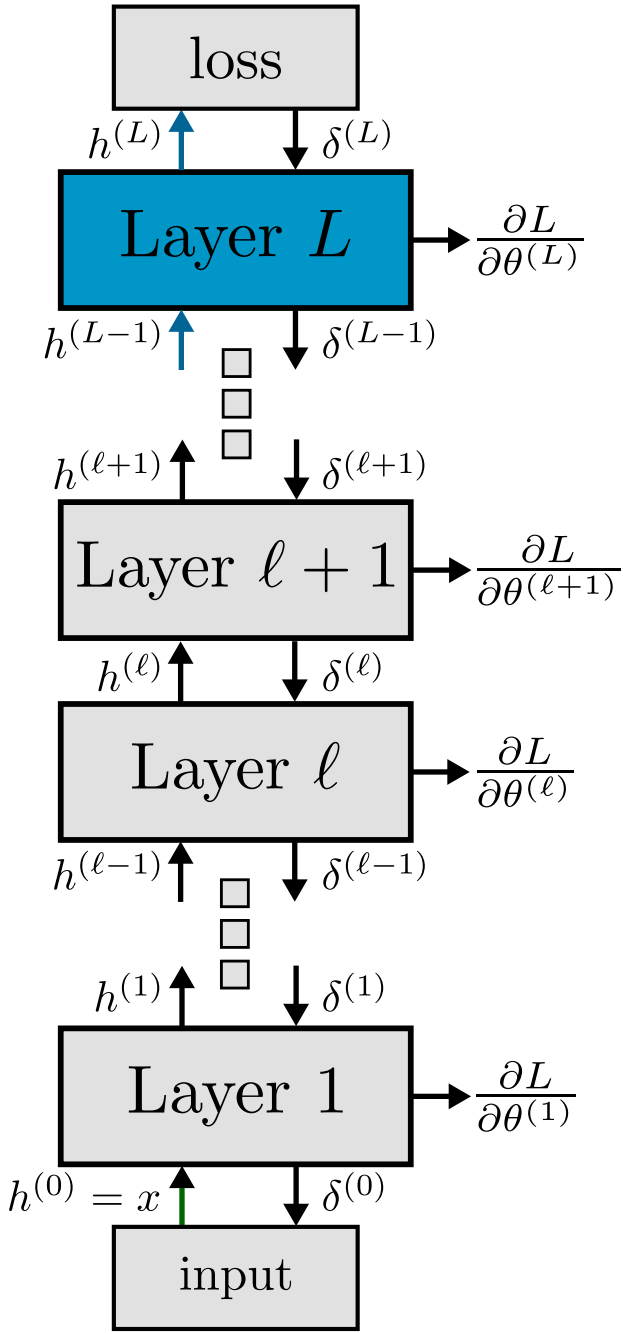
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



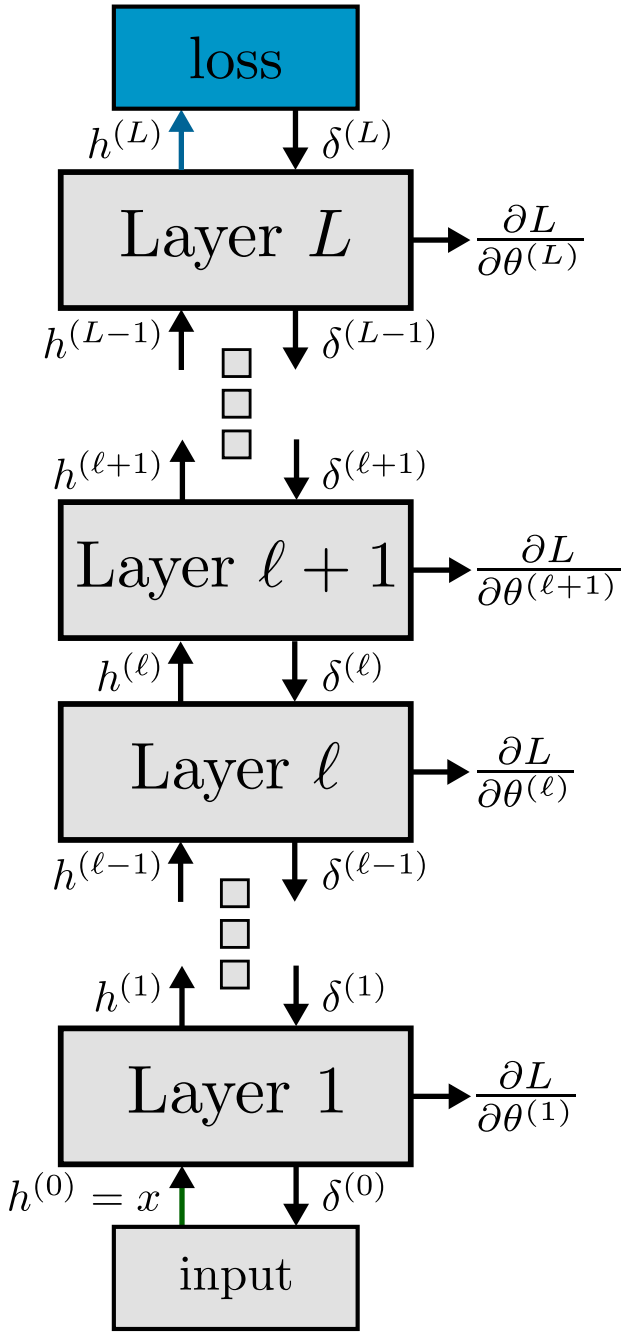
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



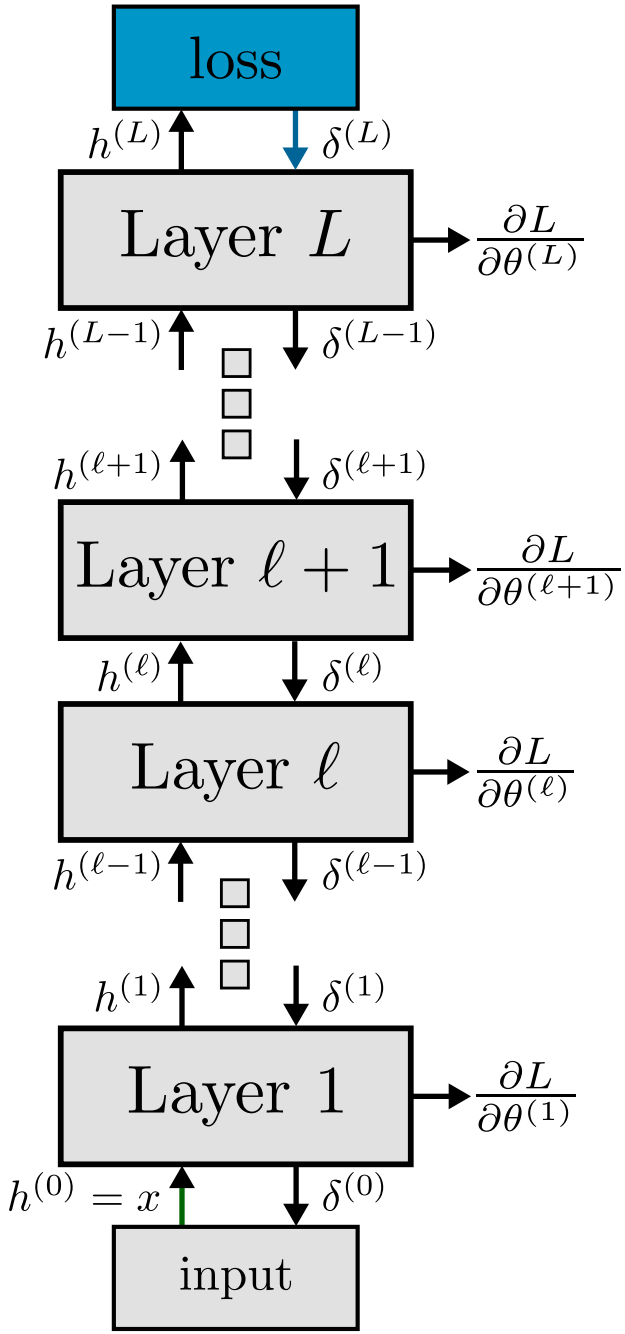
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



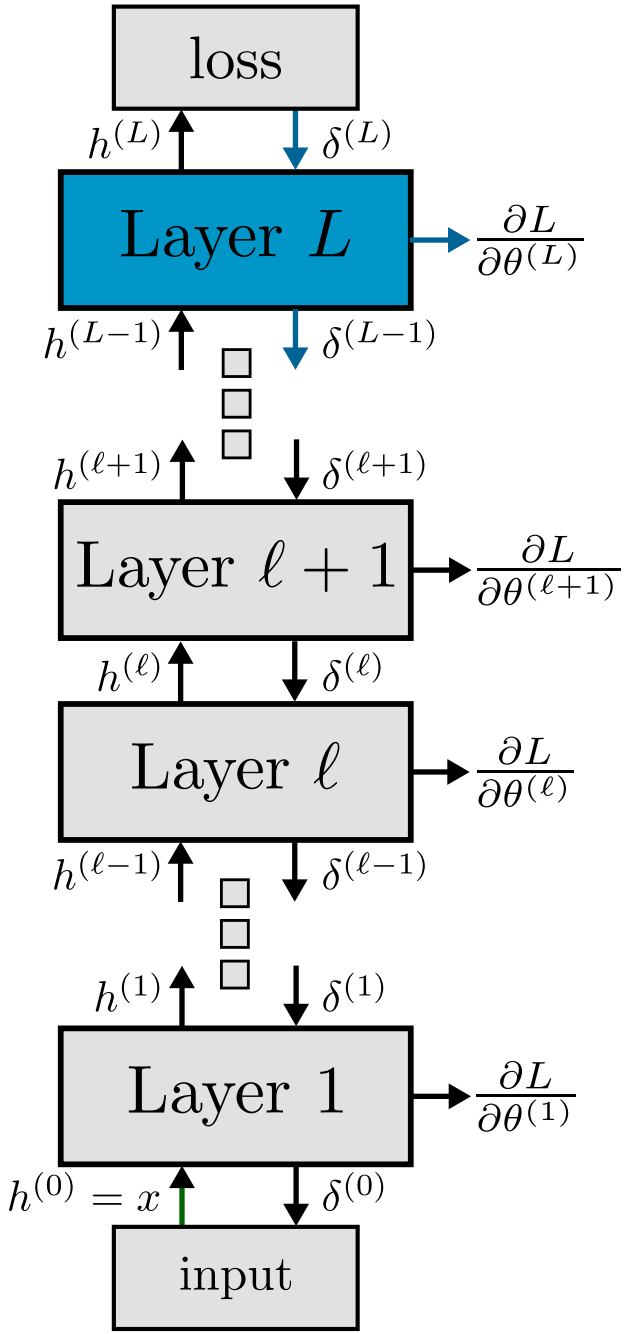
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- **Feedforward** and backpropagation are the upwards and downwards flows, respectively.



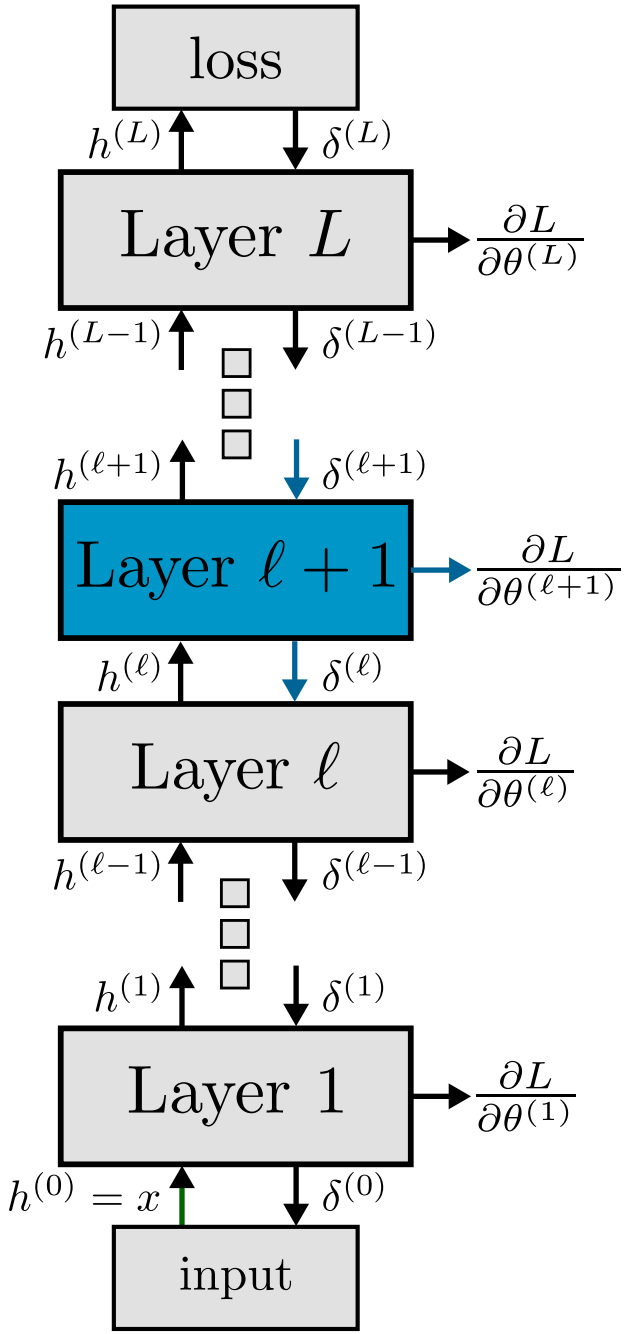
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and **backpropagation** are the upwards and downwards flows, respectively.



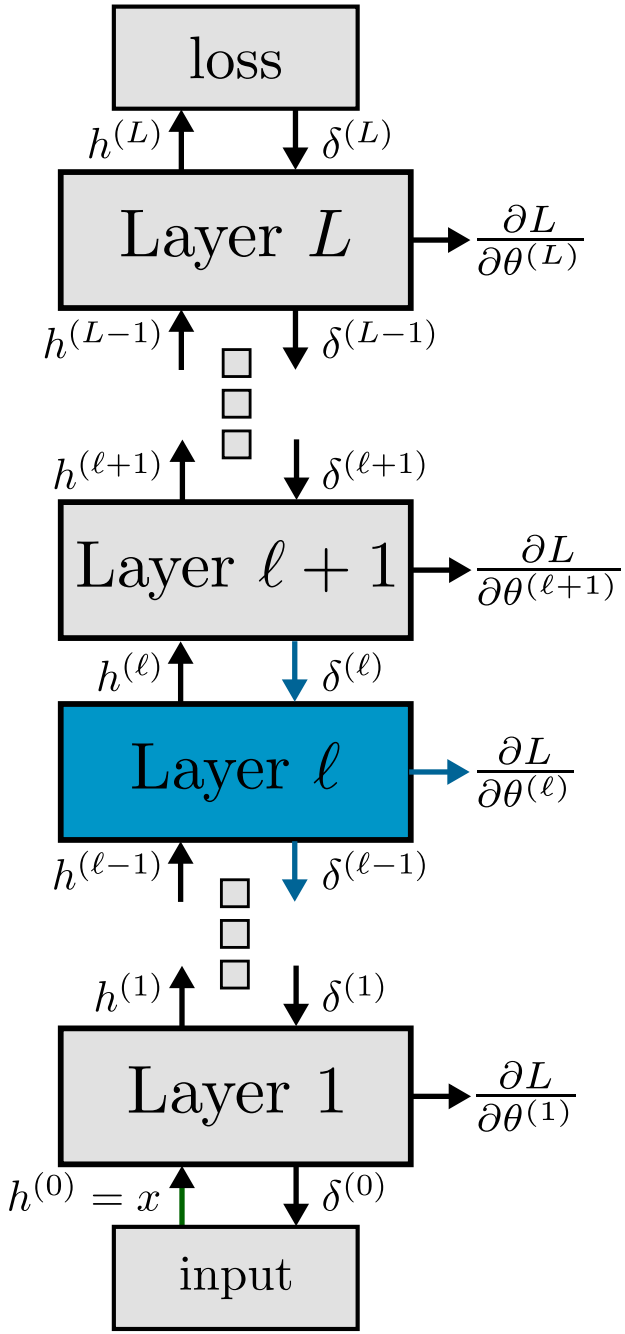
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and **backpropagation** are the upwards and downwards flows, respectively.



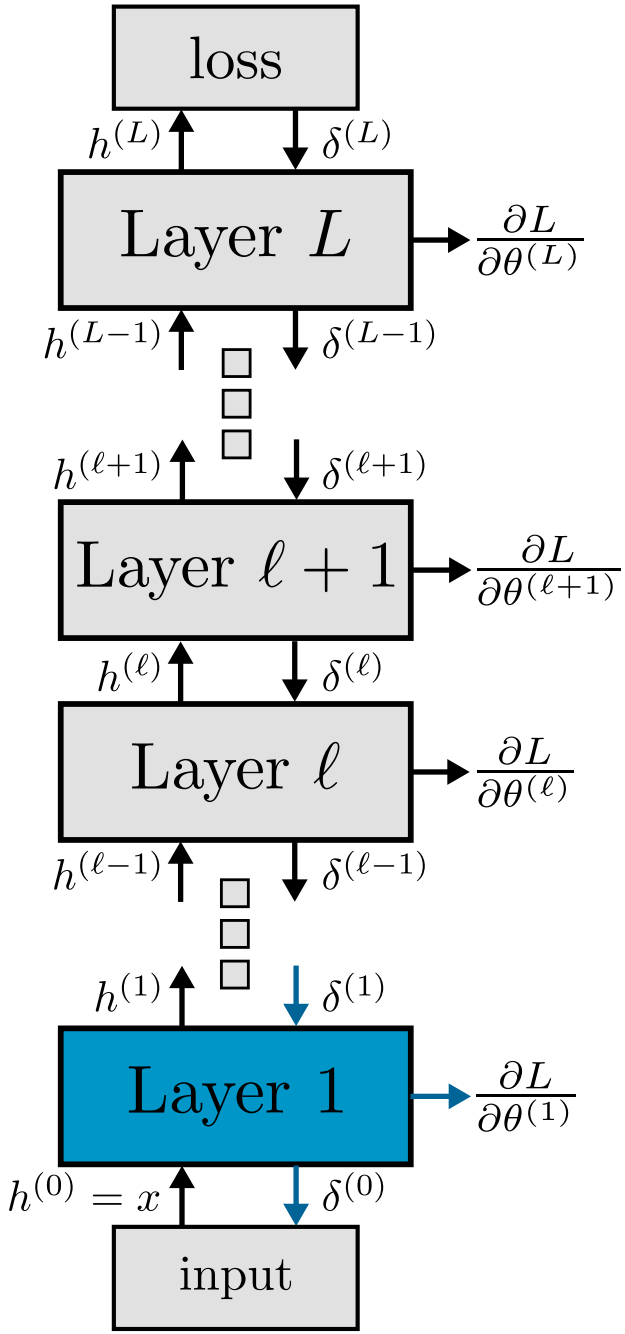
- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and **backpropagation** are the upwards and downwards flows, respectively.



- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and **backpropagation** are the upwards and downwards flows, respectively.



- Combining this ideas we can define a **Computational Graph** for our FNN model.
- This formulation makes it easier to implement and deploy in practice.
- Feedforward and **backpropagation** are the upwards and downwards flows, respectively.



Deep Learning: Software

- Writing your own neural networks from scratch is complex and error-prone.
- There are many frameworks that provide tools for designing Neural Networks in the modular form.
  - PyTorch
  - TensorFlow
  - Caffe(2)
  - Keras
  - MXNet, Theano, CNTK,...

Deep Learning: Software

- Writing your own neural networks from scratch is complex and error-prone.
- There are many frameworks that provide tools for designing Neural Networks in the modular form.
  - **PyTorch (We'll focus on this one for now)**
  - TensorFlow
  - Caffe(2)
  - Keras
  - MXNet, Theano, CNTK,...

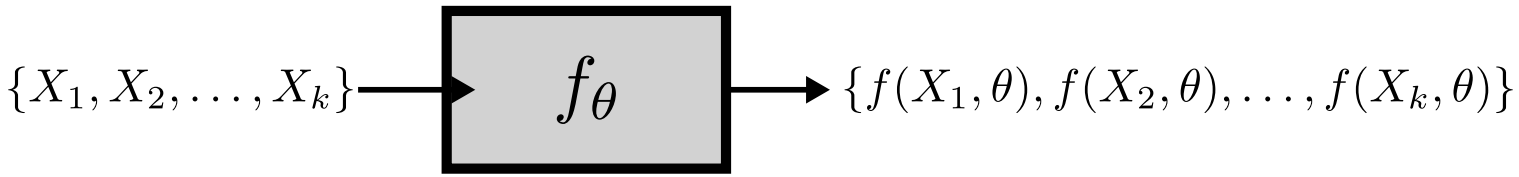
PyTorch - Introduction

Fundamental Concepts

- **Tensors:** Multidimensional arrays.
- **Autograd:** PyTorch can track your operations and automatically construct their gradients.
- **Modules:** Similar to what we discussed, it can encompass several components, like a layer or a loss function.

PyTorch - Introduction

When performing computations with batches of data, as is usually done, we want to take advantage of distributional computing. Therefore our computations we'll be done in batches of data



Therefore we usually stack our batch in tensors.

PyTorch - Introduction

Tensors:

- Tensors are a generalization of vector and matrices for higher dimensions. Computationally they are represented as $d-$dimensional arrays.
- **Example** A multi-channel image is a 3d tensor.