Feed Forward Neural Networks

Oct. 26. 2016

Guanlin Li

Outline

- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Warming Up

Super MarI/O

Warming Up

Show, attend, and tell

Warming Up

Differentiable Neural Computer

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

Task 4: Two Argument Relations
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

Task 5: Three Argument Relations
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

Task 7: Counting
Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
How many objects is Daniel holding? A: two

Task 8: Lists/Sets
Daniel picks up the football.
Daniel drops the newspaper.
Daniel picks up the milk.
John took the apple.
What is Daniel holding? milk, football

Task 9: Simple Negation
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? A:no
Is Sandra in the office? A:yes

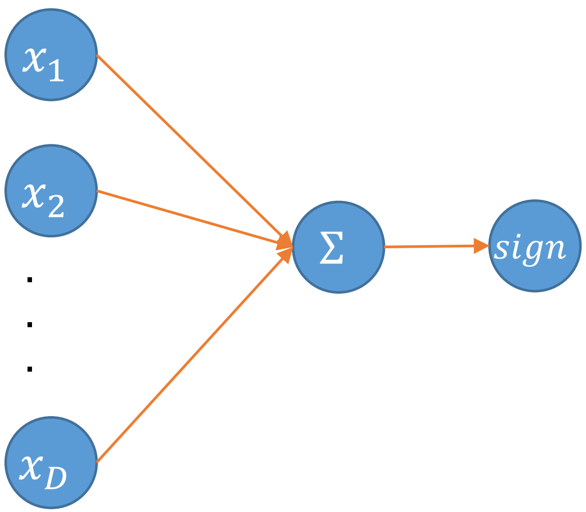
Task 10: Indefinite Knowledge
John is either in the classroom or the playground.
Sandra is in the garden.
Is John in the classroom? A:maybe
Is John in the office? A:no

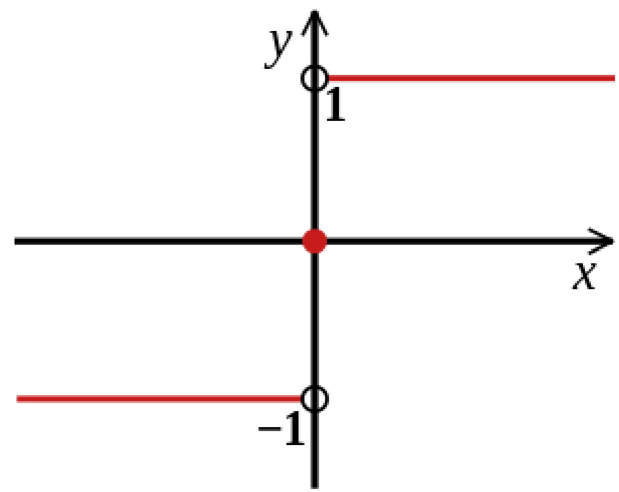
Outline

- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Building Blocks of A Feed Forward NN

- Remember: a linear binary classifier
- $a = sign(z), z = \boxed{w^T \boldsymbol{x}}$ Vector dot vector
- a is 0 or 1, depends on the value of z





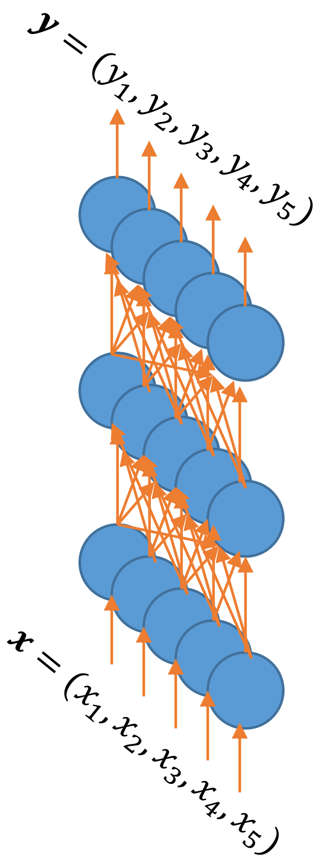
Outline

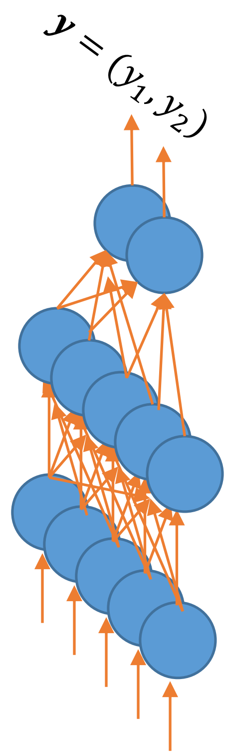
- Warming Up
- Building Blocks of A Neural Network
  - **Architecture**
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Building Blocks of A Feed Forward NN

Architecture

- What does it look like?

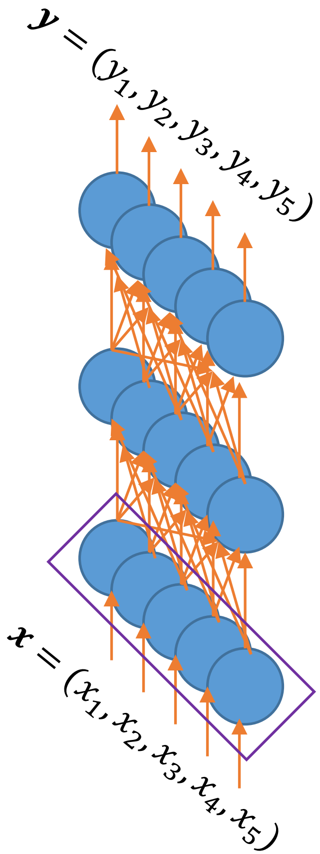


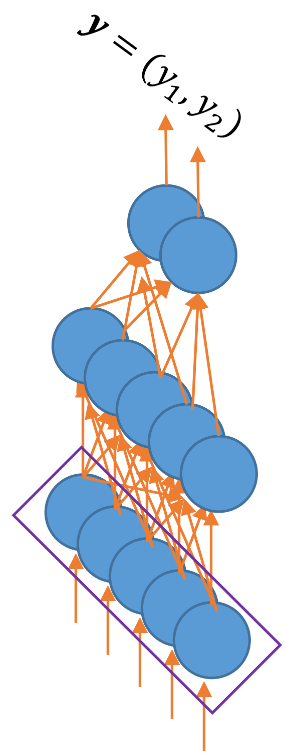


Building Blocks of A Feed Forward NN

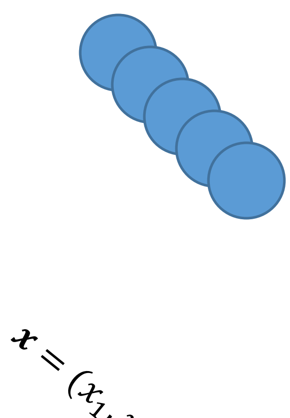
Architecture

- How does it do computation?





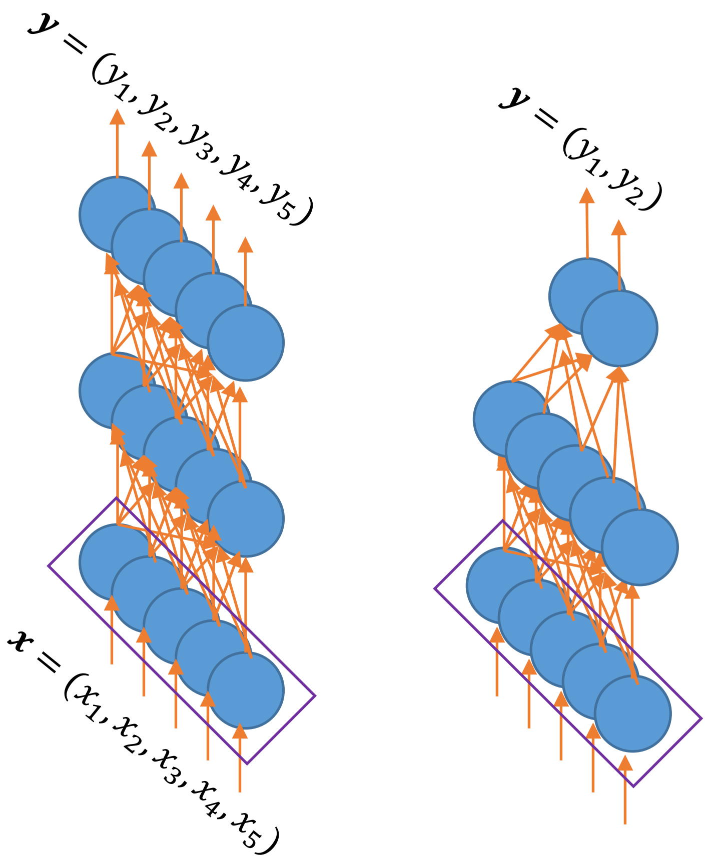
- Input: $\boldsymbol{x} = (x_1, \ldots, x_D)$
- Input Storage



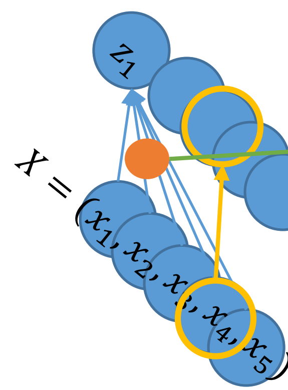
Building Blocks of A Feed Forward NN

Architecture

- How does it do computation?



- **Forward Computation:**
  - pre-activation
- $z_1 = W_{1\cdot} \cdot \boldsymbol{x}$
- $z_i = W_{i\cdot} \cdot \boldsymbol{x}$
- To vectorize:
  - $\boldsymbol{z} = W\boldsymbol{x}$

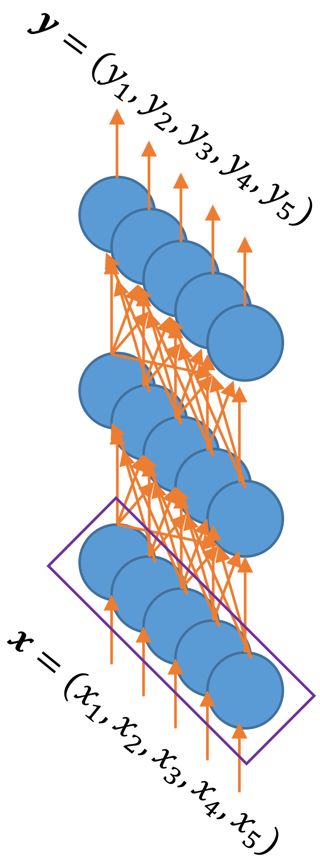


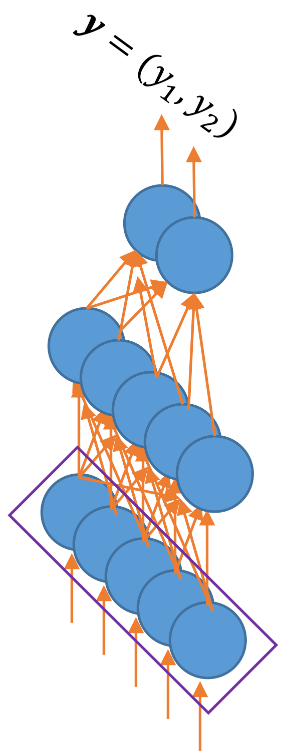
$$W = \begin{bmatrix} \boxed{w_{11} & w_{12} & w_{13} & w_{14} & w_{15}} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix}$$

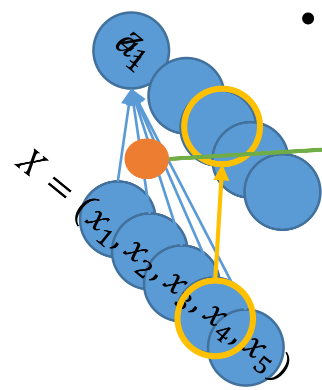
Building Blocks of A Feed Forward NN

Architecture

- How does it do computation?







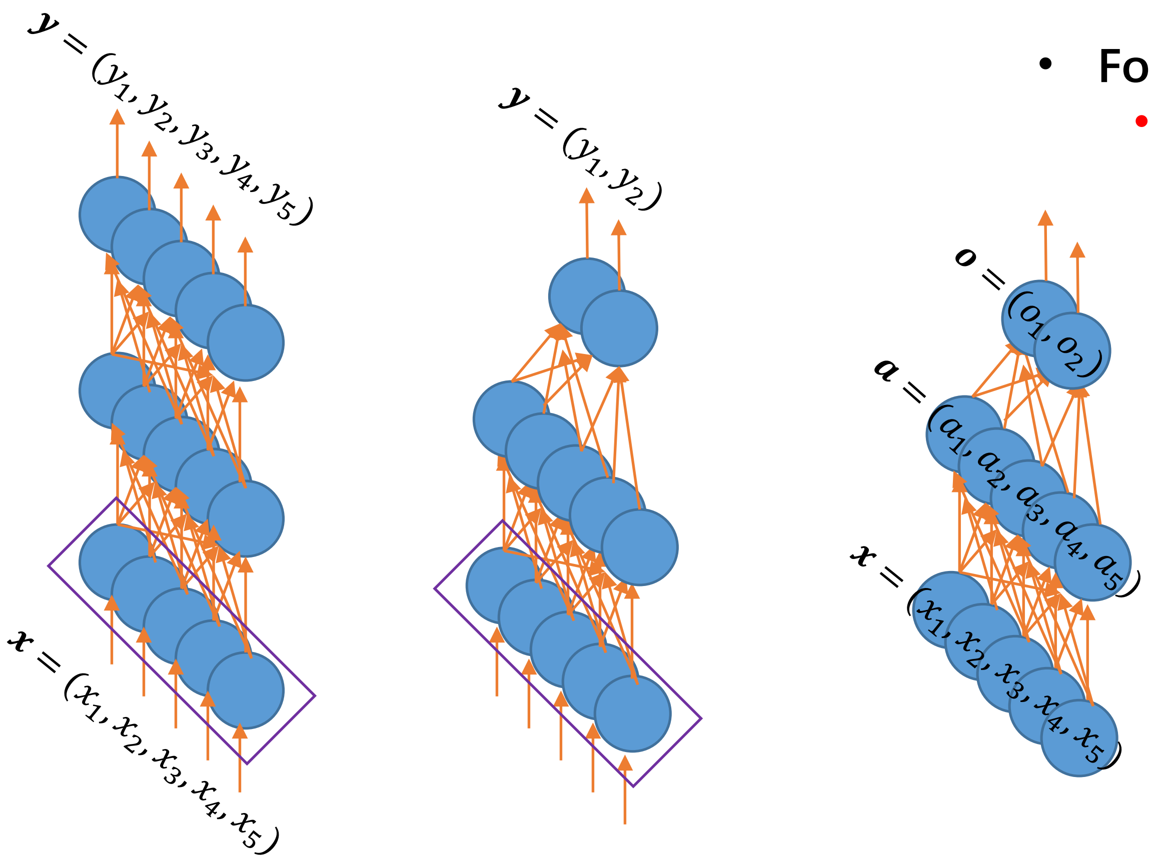
- **Forward Computation:**
  - activation
- $a_1 = f(z_1) = f(W_{1.} \cdot \boldsymbol{x})$
- $a_i = f(W_{i.} \cdot \boldsymbol{x})$
- To vectorize:
  - $f(\boldsymbol{z}) = f(W\boldsymbol{x})$ element-wise
  - $f(\boldsymbol{z}) = f(W\boldsymbol{x} + \boldsymbol{b})$

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} \end{bmatrix}$$

Building Blocks of A Feed Forward NN

Architecture

- How does it do computation?



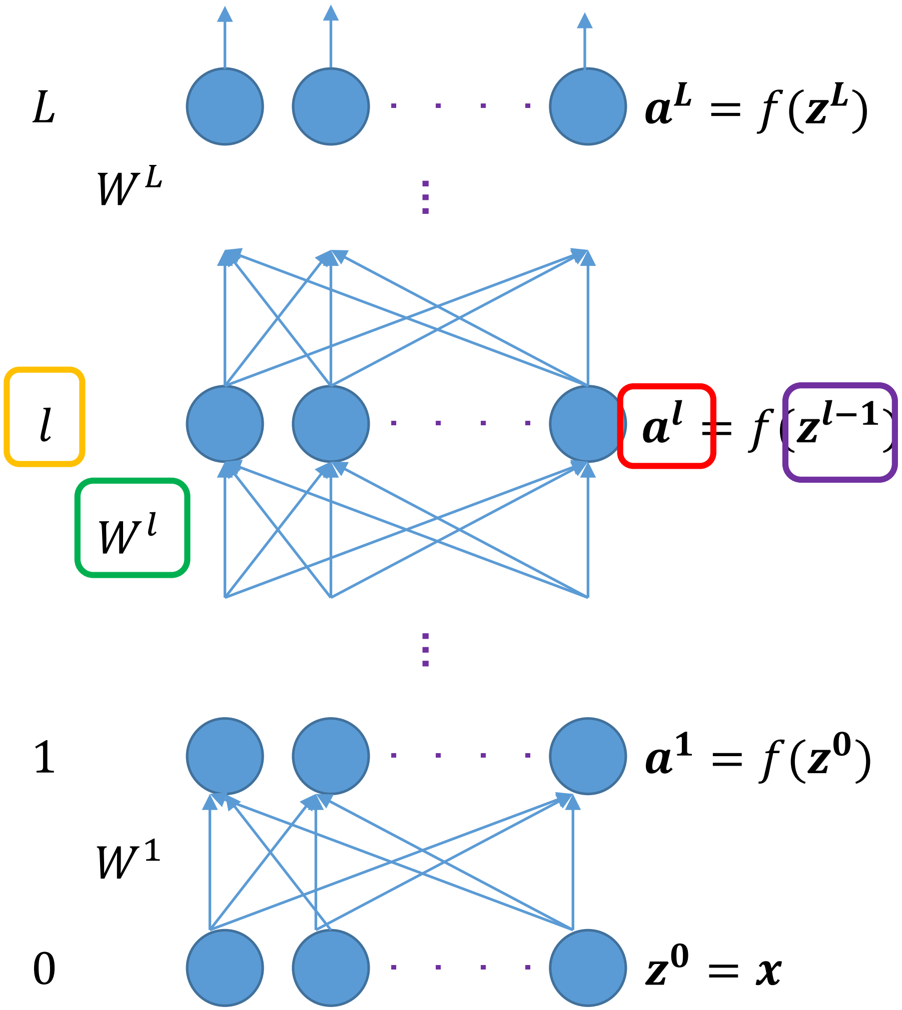
- **Forward Computation**
  - Output
    - $\boldsymbol{o} = f(\boldsymbol{W}^T \boldsymbol{a})$

Building Blocks of A Feed Forward NN

Architecture

- General architecture

- The l-th layer
- Weight matrix from $l-1$ to l-th layer
- l-th layer Pre-activation
- l-th layer activation



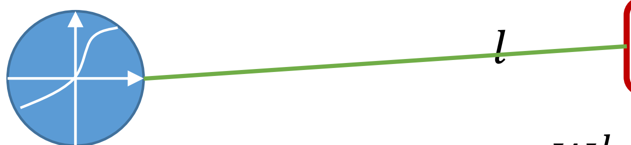
Building Blocks of A Feed Forward NN

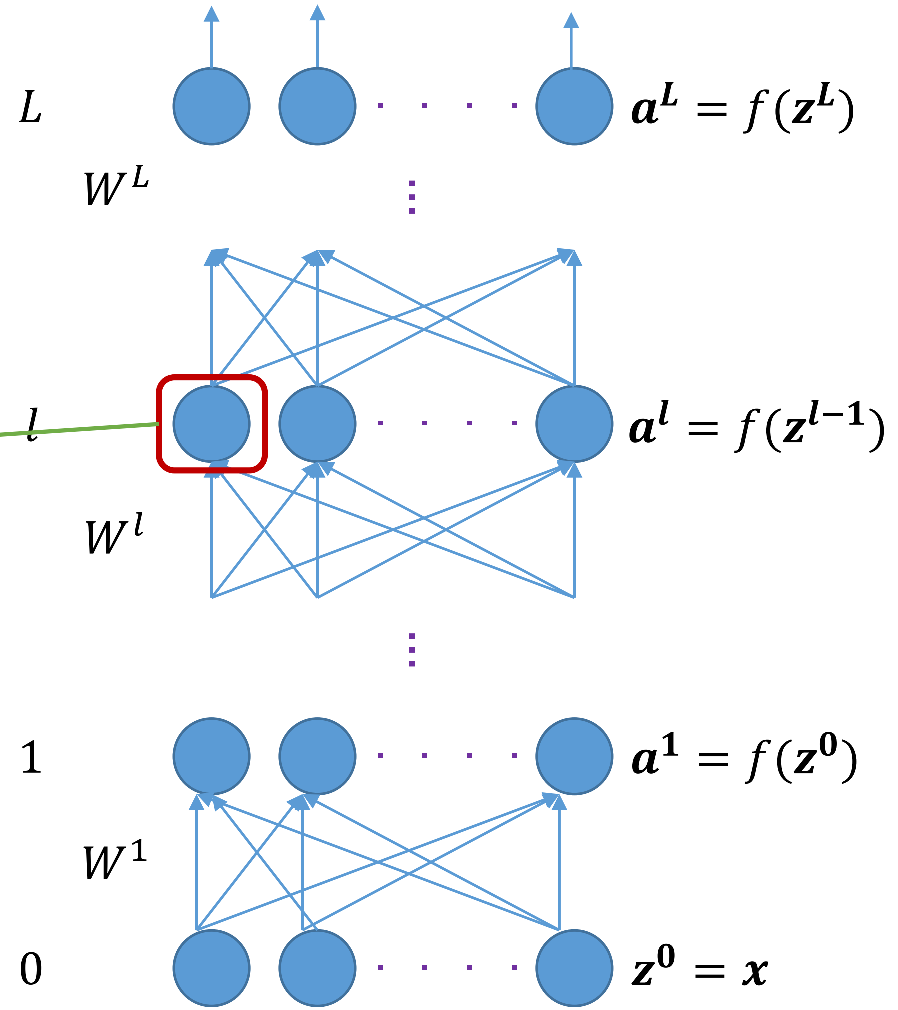
Architecture

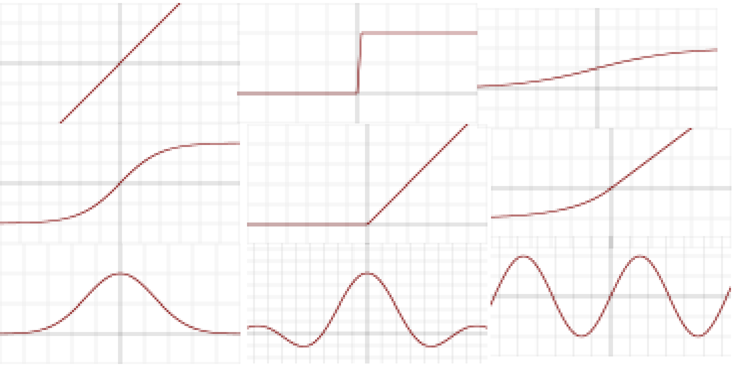
- General architecture

- Squashing/activation function

A neuron





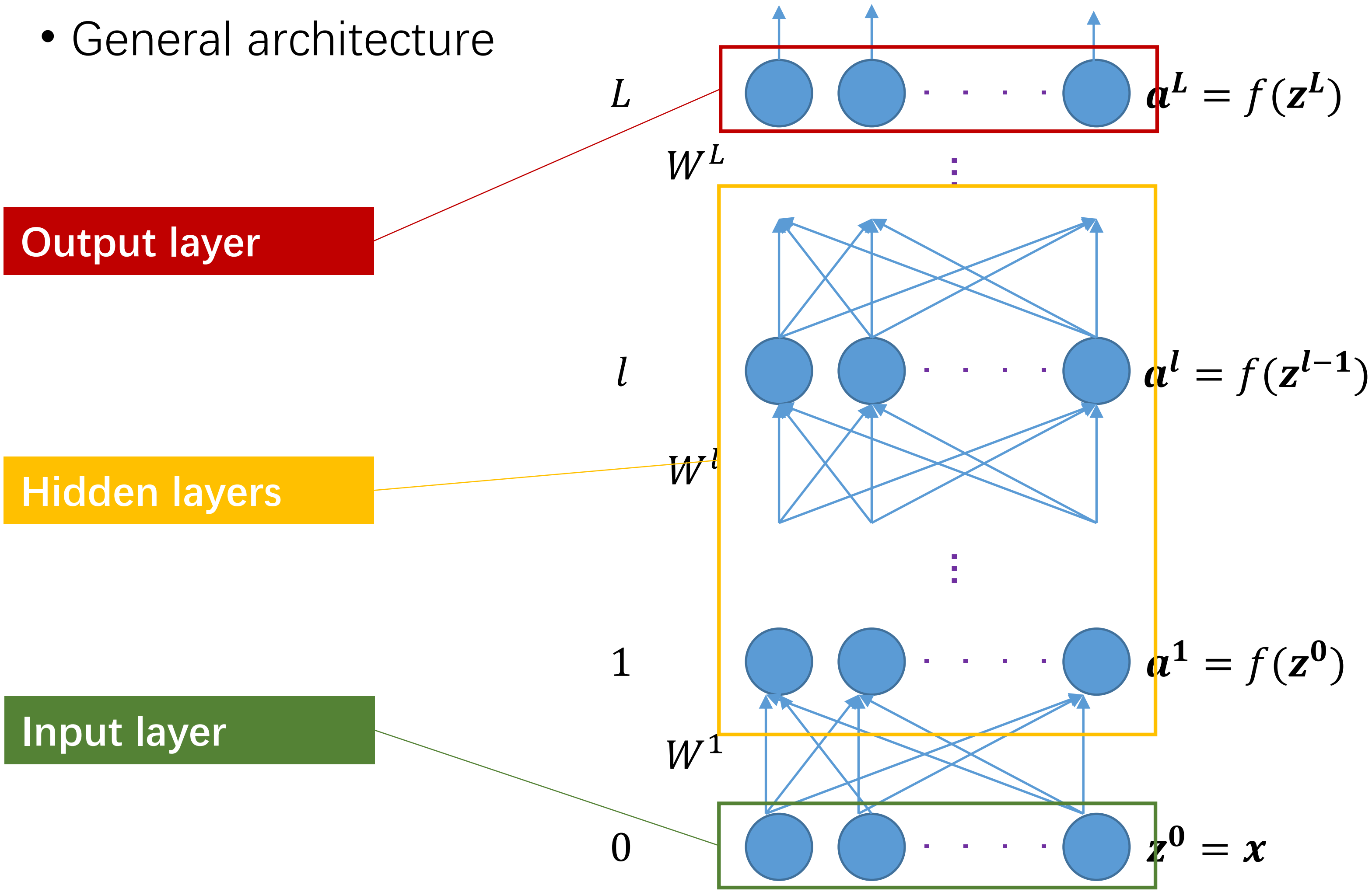


Different activation functions: https://en.wikipedia.org/wiki/Activation_function

Building Blocks of A Feed Forward NN

Architecture

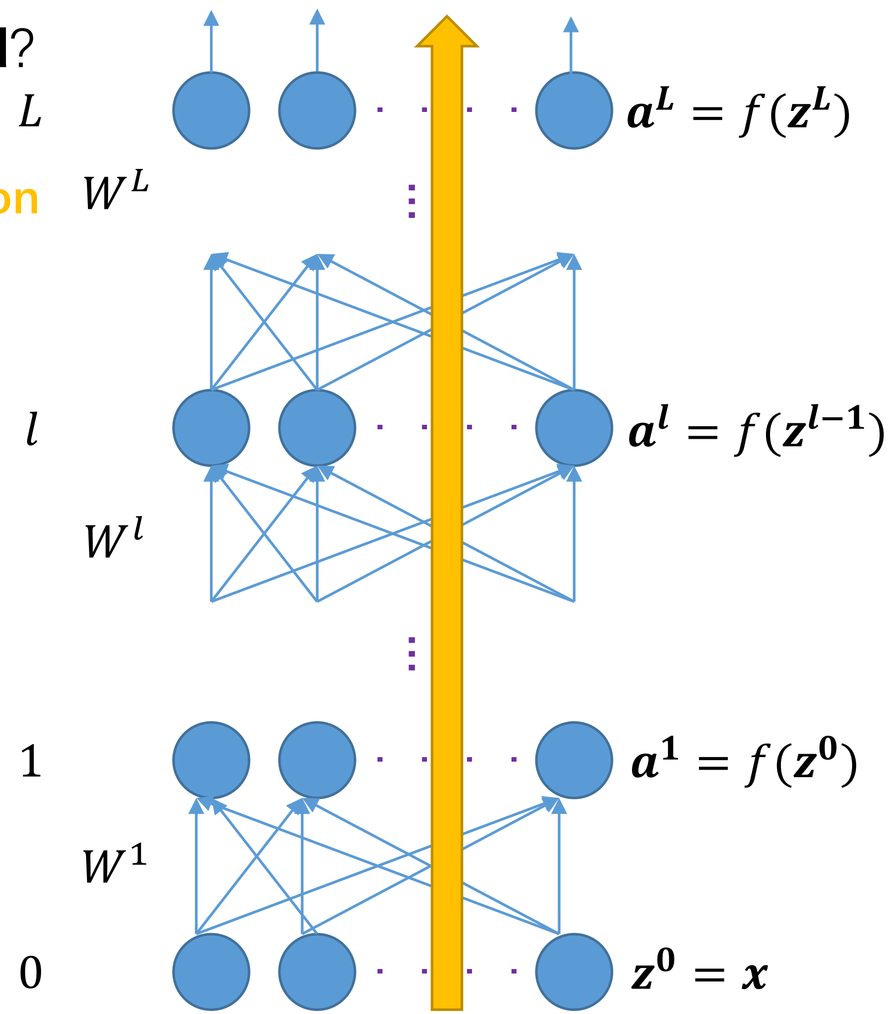
- General architecture



Building Blocks of A Feed Forward NN

Architecture

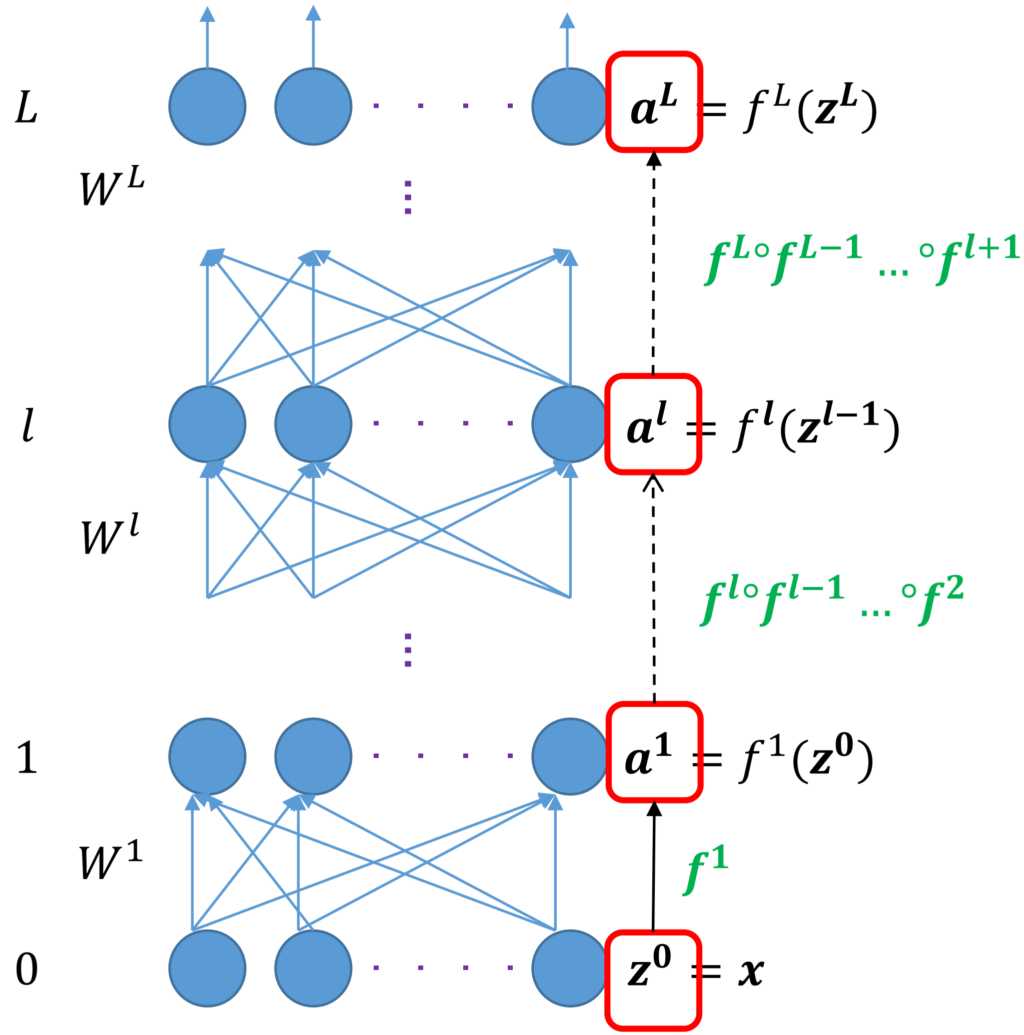
- Why call it **Feed Forward**?
  - Information/Computation
  - flow from bottom up



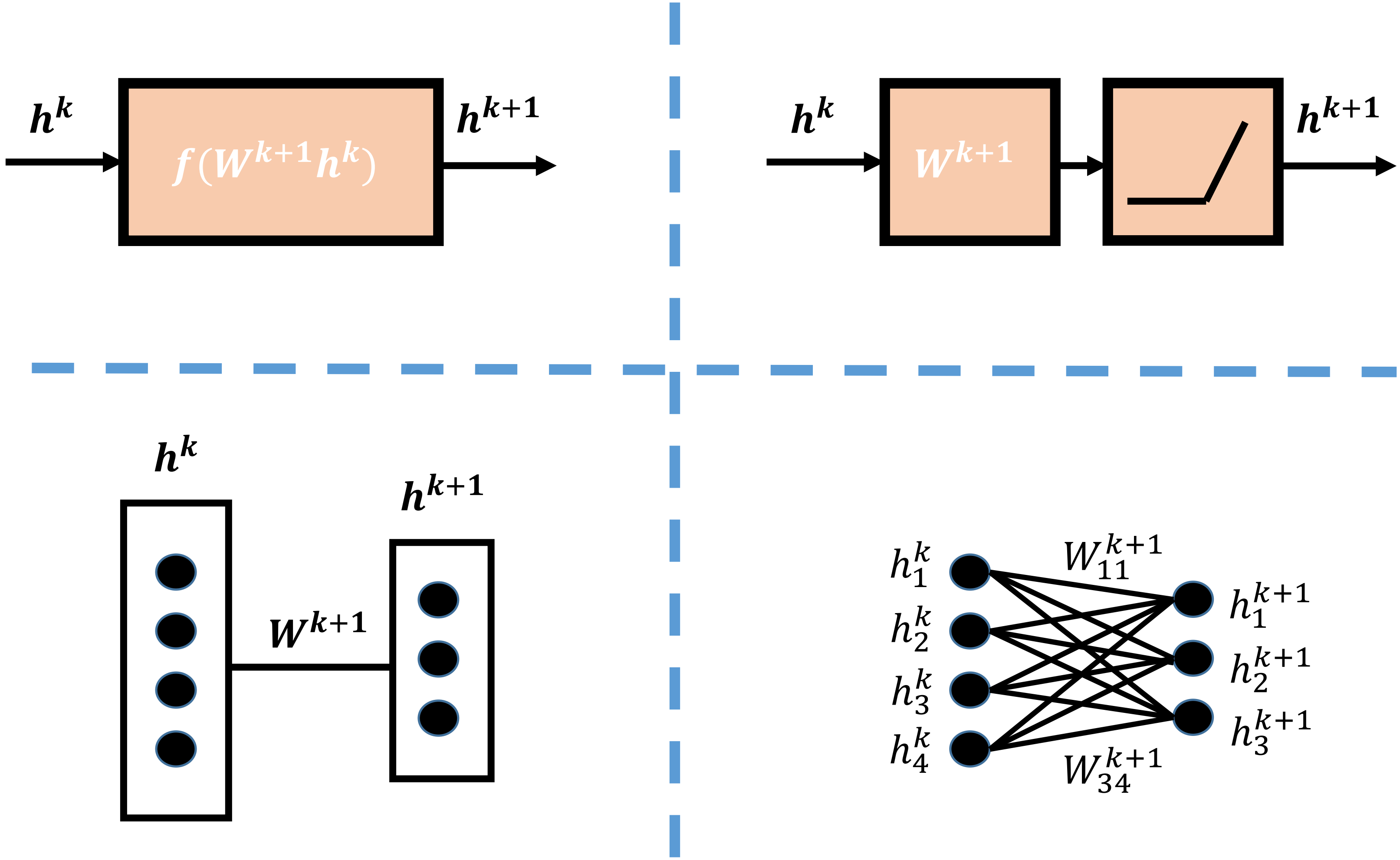
Building Blocks of A Feed Forward NN

Architecture

- Why call it **Network**?
  - Function **composition**
    - The concept of
      - Layer & Depth



Bonus: Alternative Graphical Representations of NNs



Outline

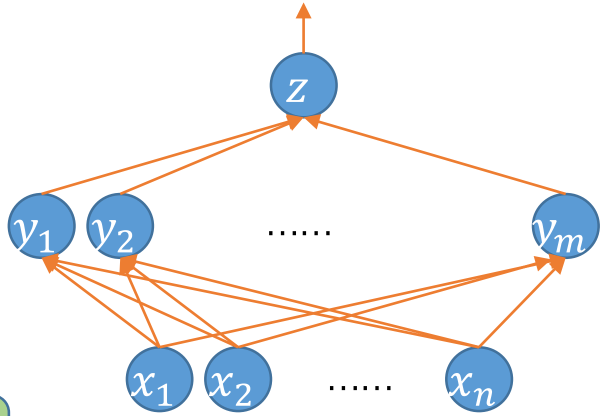
- Warming Up
- Building Blocks of A Neural Network
  - **Architecture**
  - **Functionality**
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Building Blocks of A Feed Forward NN

Functionality

- A Universal Function Approximator
  - In English: There is a single hidden layer feedforward network that approximates any measurable function to any desired degree of accuracy on some compact set K.
  - In Math: For every function g in M^r, there is a compact subset K of R^r and an $f \in \Sigma^r(\Psi)$ such that for any $\epsilon > 0$ we have $\mu(K) < 1 - \epsilon$ and for every $X \in K$ we have $|f(x) - g(x)| < \epsilon$, regardless of Ψ, r or μ.

- Idea of the proof
  - A continuous function on a compact set can be approximated by a piecewise constant function.
  - A piecewise constant function can be represented as a neural nets.



This result can be extended to approximating a function in R^D

The original paper: http://deeplearning.cs.cmu.edu/pdfs/Kornick_et_al.pdf
Michael Nielsen's online chapter: http://neuralnetworksanddeeplearning.com/chap4.html

Building Blocks of A Feed Forward NN

Functionality

- Output of a FFNN is used for prediction
  - Both regression and classification can be tackled
  - Carefully designed Output Units

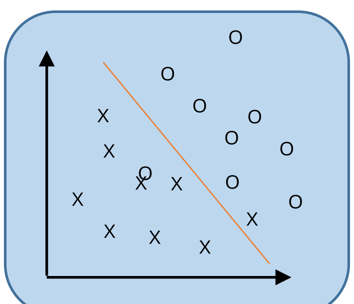
Regression	Classification

Building Blocks of A Feed Forward NN

Functionality

- The way to Neural Networks

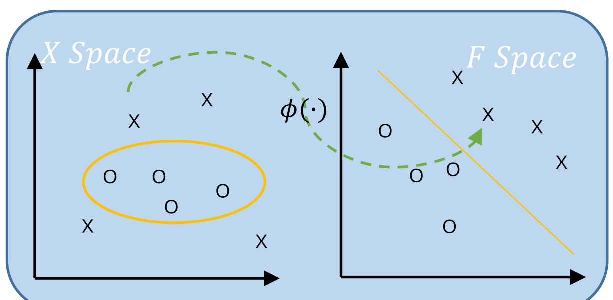
- Linear in weights and features
- Draw decision boundary in original feature space



$f(x) = w^T \boldsymbol{x}$

Linear Model (LM)

- Linear in weights but nonlinear in features
- A designed new feature space for original features to be mapped to
- Fixed mapping functions designed by heuristics



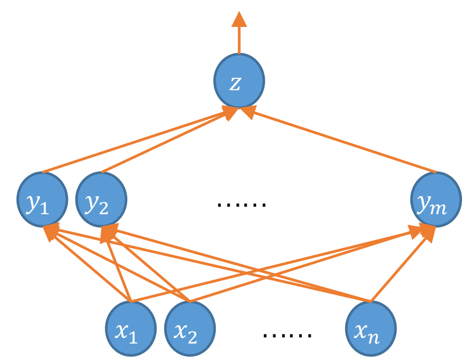
$f(x) = w^T \phi(\boldsymbol{x})$

Generalized LM

$f(x) = \sum \alpha_i K(x_i, x)$

Kernel Methods

- Both nonlinear in features and weights
- Learnable mapping from original feature space to new
- But nonlinearity gives nonconvexity



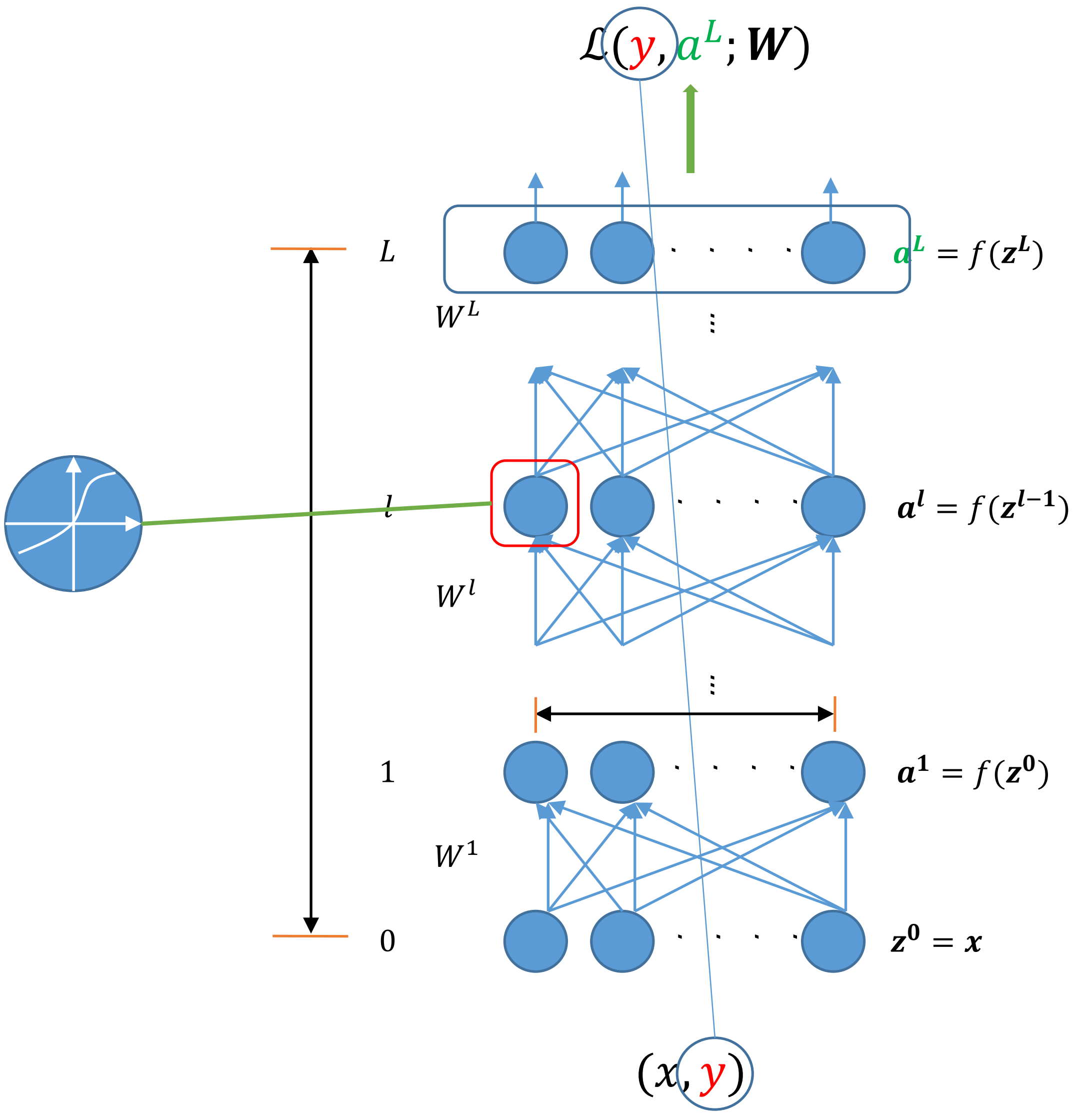
Outline

- Warming Up
- Building Blocks of A Neural Network
  - **Architecture**
  - **Functionality**
  - **Design**
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Building Blocks of A Feed Forward NN

Design

- Cost/Objective function - Supervised
- The output layer
- The squashing function
- The width and depth



Building Blocks of A Feed Forward NN

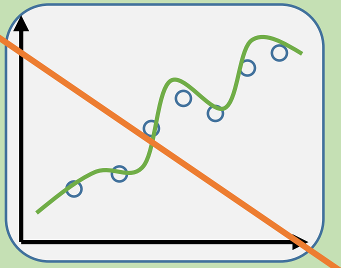
Design – Cost Function

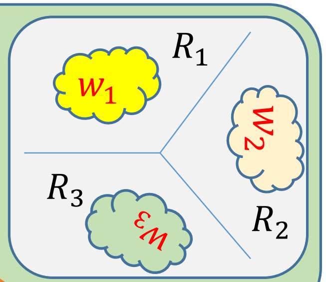
- Cost(lost)/objective function – Supervised $(\boldsymbol{x}, \boldsymbol{y})$
  - Cost function proposes an optimization problem
  - It is designed according to our prior knowledge, some universal principle, e.g. Diversity/Error, Maximum Likelihood, Maximum Posterior etc.
  - Different models tend to have different objectives

However, at least **two ways** of thinking of designing an **machine learning model**

Un-probabilistic

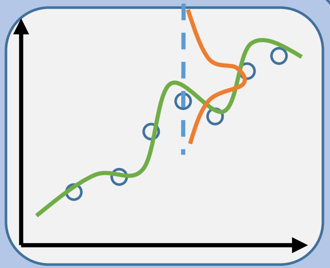
- Regression: e.g. polynomial regression
  - $y = polynomial(\boldsymbol{x}, w)$
- Classification: e.g. linear discriminant
  - $y_i = w_i^T \boldsymbol{\phi}(\boldsymbol{x})$, i indexes class
  - Decision rule: which y_i is the largest





Probabilistic

- Regression & Classification
  - Instead of $f: \boldsymbol{x} \to y$ directly
  - Model $y|\boldsymbol{x} \sim \Pr(y|\boldsymbol{x})$
- **Neural nets can approach both of them**
- **Probabilistic takes over mostly**

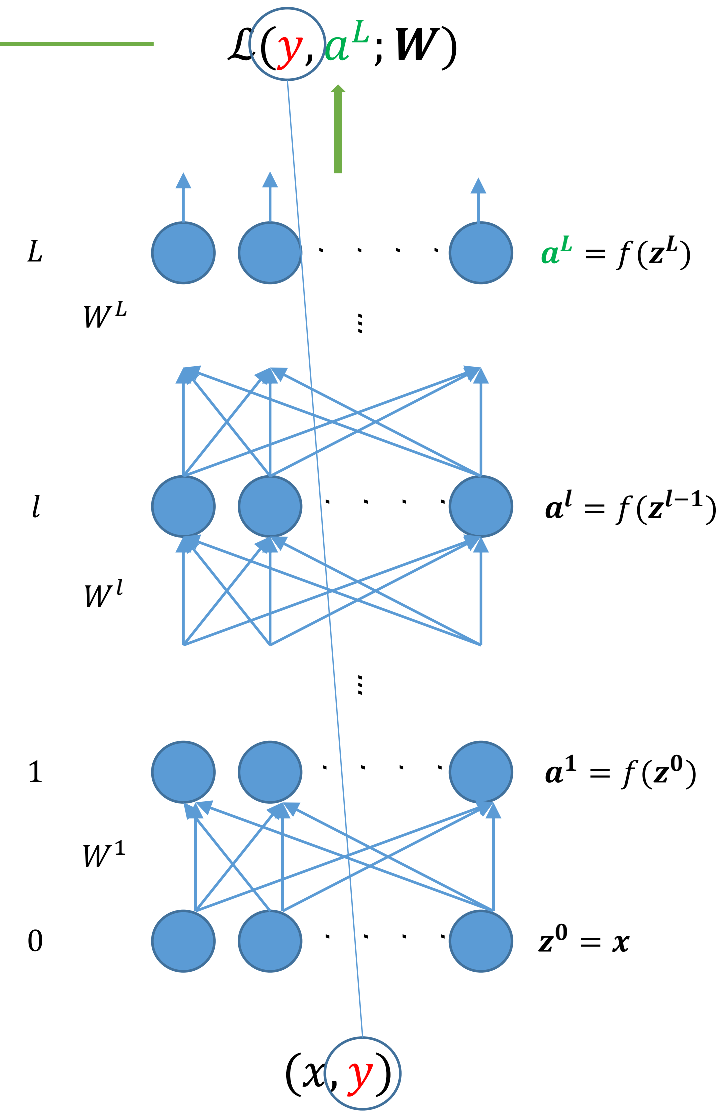


Principles

Building Blocks of A Feed Forward NN

Design – Cost Function

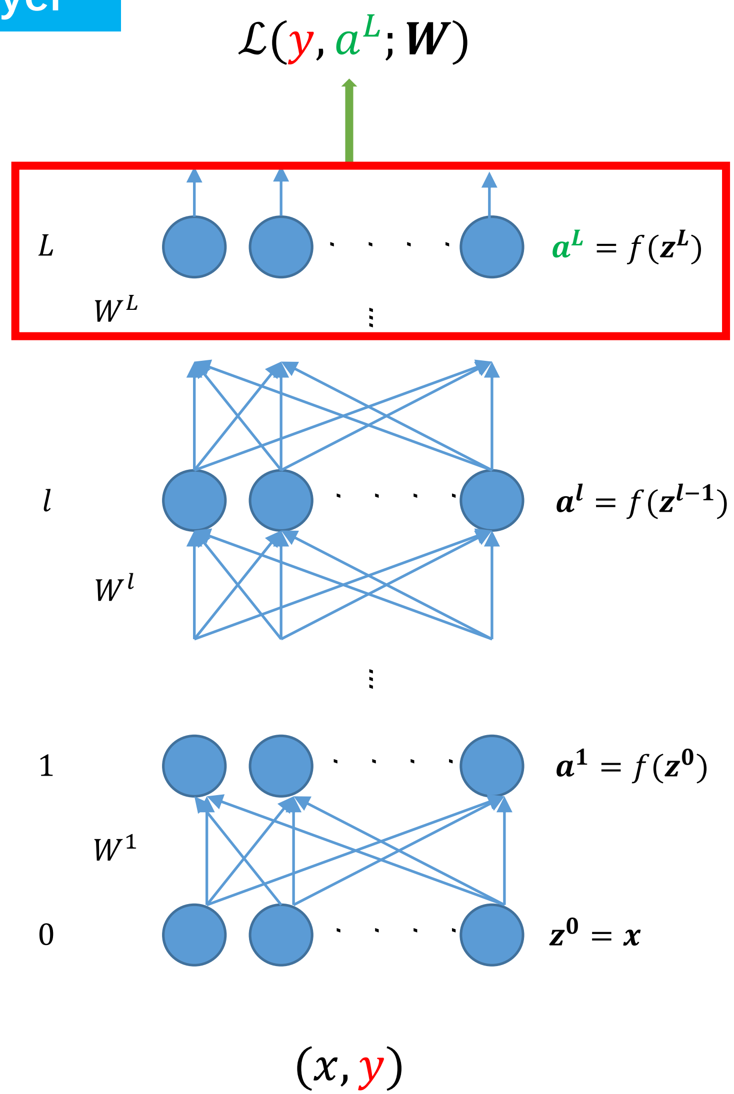
- Cost function
  - *"Most modern neural networks are trained using maximum likelihood, equivalently described as cross-entropy between the training data and the model distribution. This cost function is given by:* $J(\theta) = - E_{\boldsymbol{x},y\sim\hat{P}_{data}} log P_{model}(y|\boldsymbol{x})$ *"*
- This means our model is designed to describe the conditional probability of regressed value/class label given $\boldsymbol{x}$



$$\boxed{\begin{aligned}&\mathcal{L}(y, a^L; W) \\ &= -E_{\boldsymbol{x},y\sim\hat{P}_{data}} log P_{model}(y|\boldsymbol{x})\end{aligned}}$$

Building Blocks of A Feed Forward NN

Design – Output Layer



Design consideration

- The functional form of f
- The unit number

Design philosophy

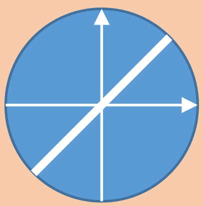
- Task specification
  - Regression or classification
  - The class number

Building Blocks of A Feed Forward NN

Design – Output Layer

- Output layer is tightly bound with the form of cost function

Linear Units for Gaussian Distribution



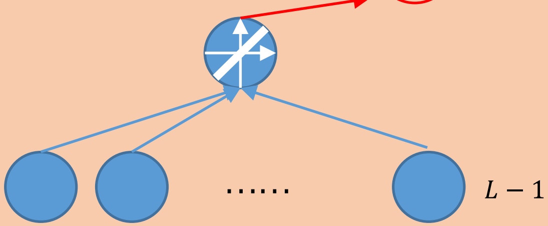
$\hat{y} = W^{T}_{D^{L-1}\times 1}\boldsymbol{a}^{L-1}$

mean

Gaussian Distribution:

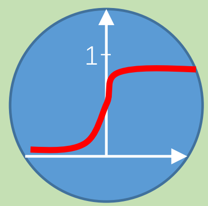
$$p(y|\boldsymbol{x}) = \boldsymbol{N}(y;\hat{y},\boldsymbol{I})$$

?



A Regression Model

Sigmoid Units for Bernoulli Distribution

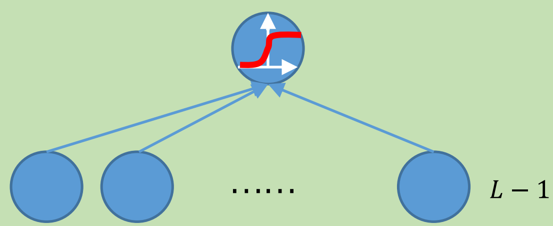


$$\boldsymbol{z}^{\boldsymbol{L}} = W_{\cdot i}^{L^{T}}\,\boldsymbol{a}^{L-1}$$

$$a^{L} = \sigma(\boldsymbol{z}^{L})$$

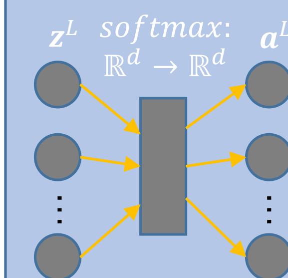
Bernoulli Distribution:

$$p(y=1|\boldsymbol{x}) = a^{L} = \sigma(\boldsymbol{z}^{L})$$



A Binary Classification Model

Softmax Units for Multinoulli Distribution



$$a_i^{L} = softmax(\boldsymbol{z}^{L})_i = \frac{\exp(z_i^{L})}{\Sigma\exp(z_j^{L})}$$

$$\boldsymbol{a}^{L} = softmax(\boldsymbol{z}^{L})$$

Multinoulli Distribution:

$$(p_1,\ldots,p_C|\boldsymbol{x}) = \boldsymbol{a}^{L} = softmax(\boldsymbol{z}^{L})$$

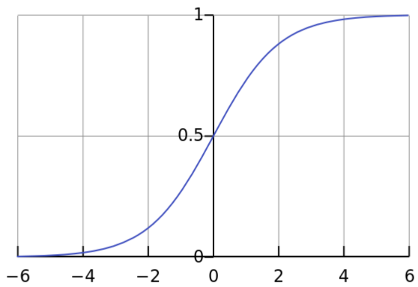
Where p_i is the probability that $\boldsymbol{x}$ belongs to the i-th class

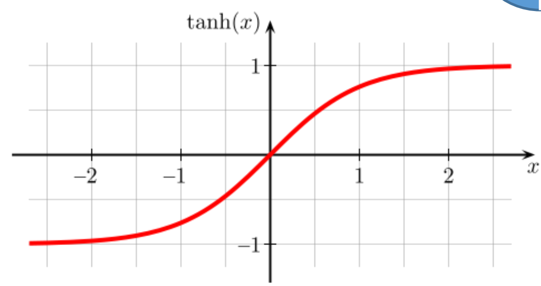
A Multiple Classification Model

Building Blocks of A Feed Forward NN

Design – Squashing Function

- Sigmoid/Hyperbolic Tagent

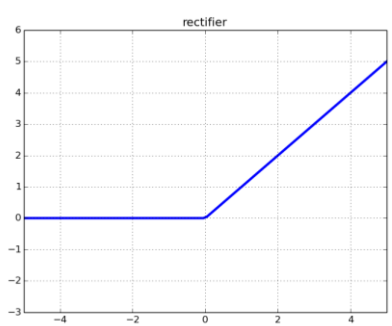




$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$

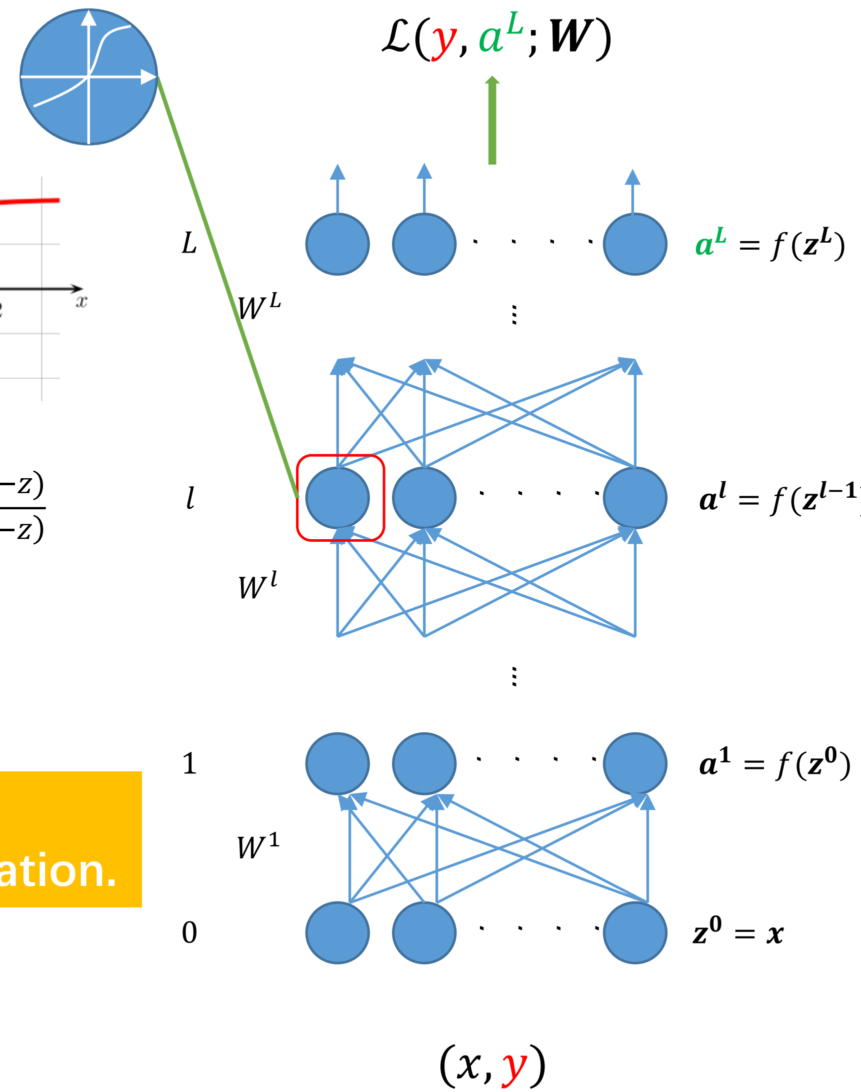
- ReLU



$$f(z) = \max(0, z)$$

New Concept:

The problem of saturation.



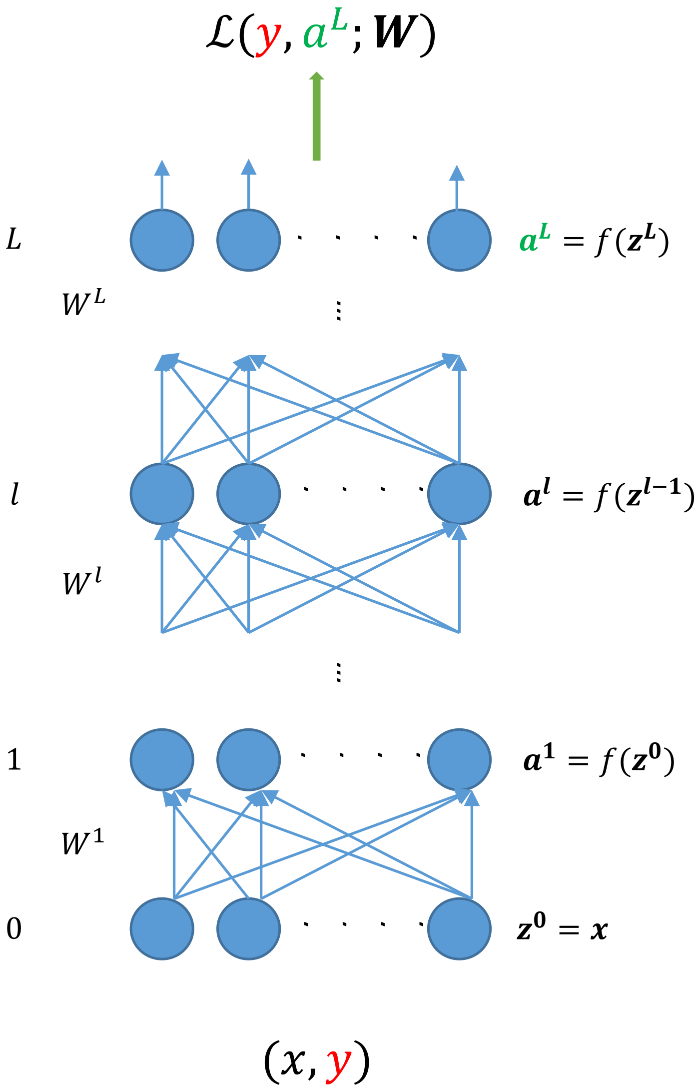
Outline

- Warming Up
- Building Blocks of A Neural Network
  - **Architecture**
  - **Functionality**
  - **Design**
- Back Propagation: A Nice Explanation
  - **Matrix Form, Computational Graph**
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Back Propagation: A Nice Explain

Optimization Problem

- Our cost function
  - $\mathcal{L}(y, a^L; \boldsymbol{W})$ where a^L is function of a^{L-1}, and a^L is function of a^{L-1},⋯
- Gradient-based Learning
  - The parameters of our model: $\boldsymbol{W}$
  - We should compute the gradient of the cost with respect to (w.r.t.) $\boldsymbol{W}$
  - $\boldsymbol{W}$ is stratified/layer-by-layer, W^l is the l-th layer weight matrix
  - Generally we should derive: $\partial\mathcal{L}/\partial W_{ij}^l$
  - Then, we can come to Gradient Descent to learn a suitable $\boldsymbol{W}$
- Question: HOW?



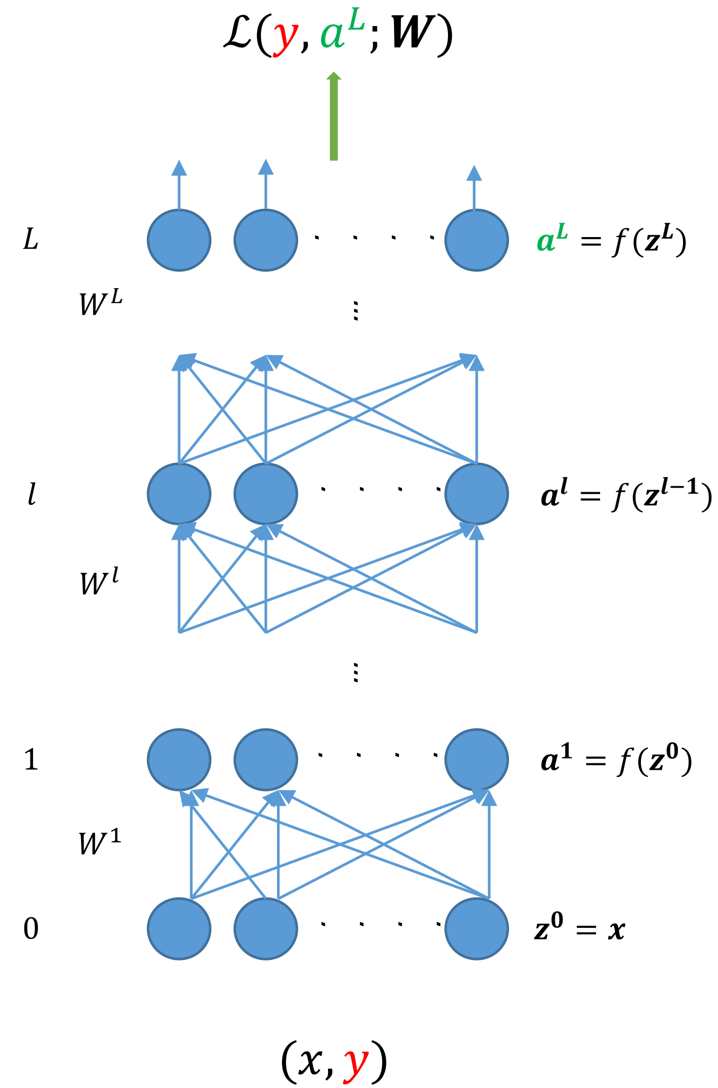
Back Propagation: A Nice Explain

Gradient Computation

- **Our target**: to compute $\partial\mathcal{L}(\boldsymbol{W})/\partial W_{ij}^{l}$ in *analytic forms* or a *procedural way*.

- Mathematically speaking, $\mathcal{L}$: $\mathbb{R}^{|\boldsymbol{W}|} \rightarrow \mathbb{R}$ is a composite function, the composition could be seen as:
  - Layer-by-layer, or
  - Neuron-after-neuron

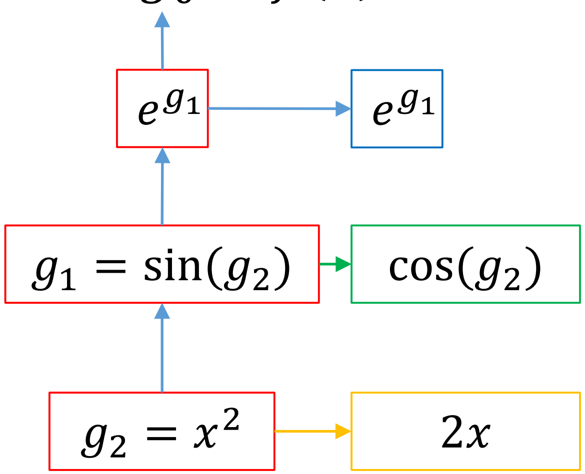
- Composite function: is the composite of some elementary functions, such as:
  - $\sin(x)$, x^p, e^x, $\log_a x$, etc.



Back Propagation: A Nice Explain

Gradient Computation

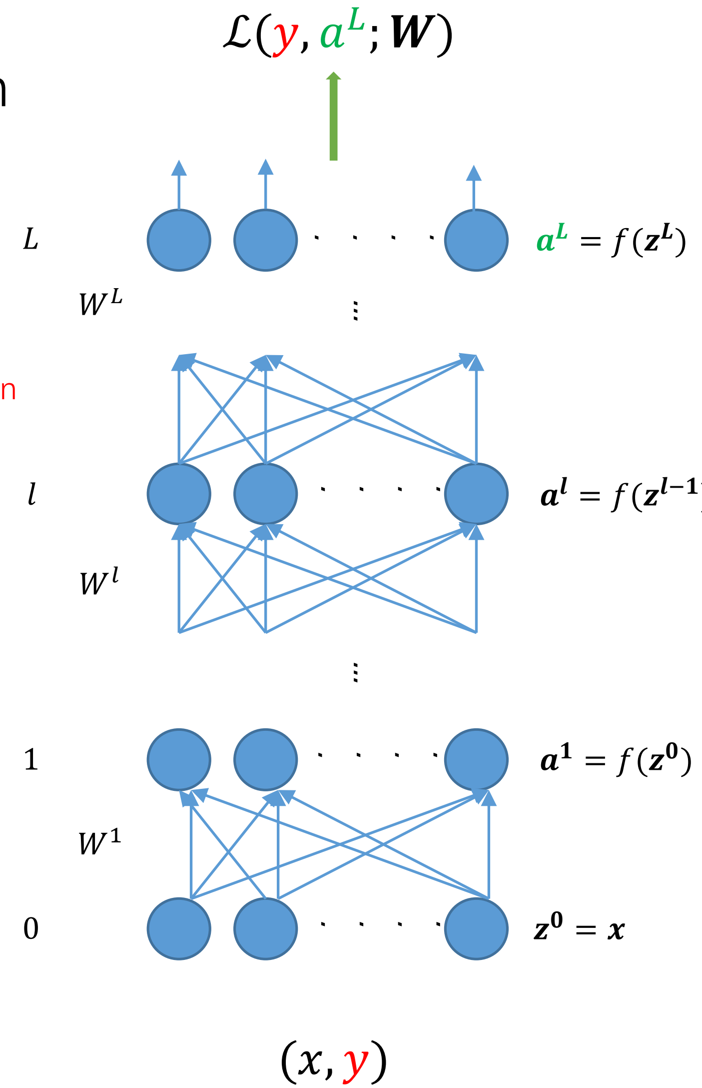
- So let us review a basic gradient derivation of a concrete composite function.
- $g_0 = f(x) = e^{\sin(x^2)}$



$$\frac{\partial f}{\partial x} = \frac{\partial(g_0 \circ g_1 \circ g_2)}{\partial x}$$

Chain rule

$$\frac{\partial g_0}{\partial g_1} \cdot \frac{\partial g_1}{\partial g_2} \cdot \frac{\partial g_2}{\partial x}$$



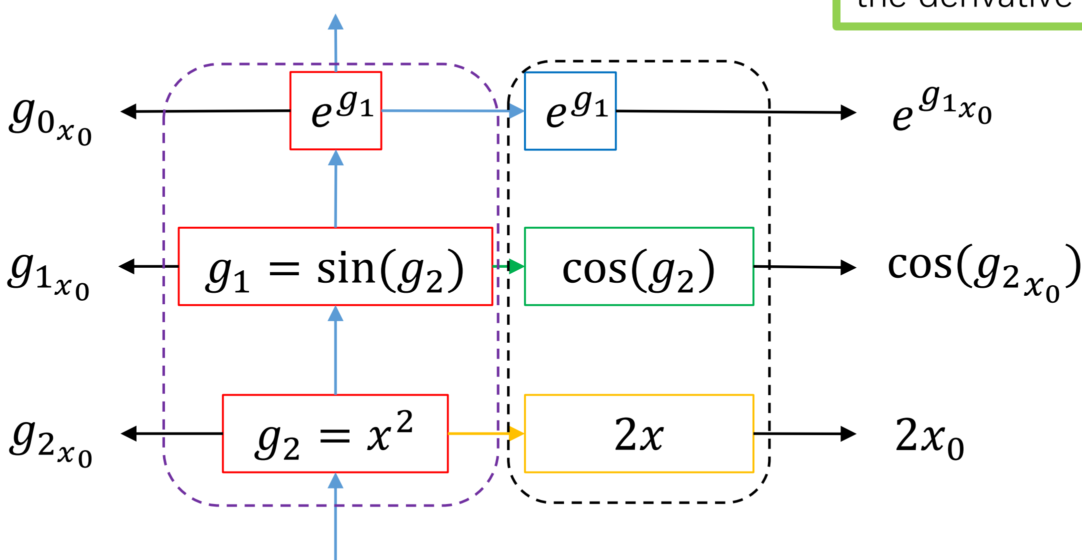
- So how we program it?

Back Propagation: A Nice Explain

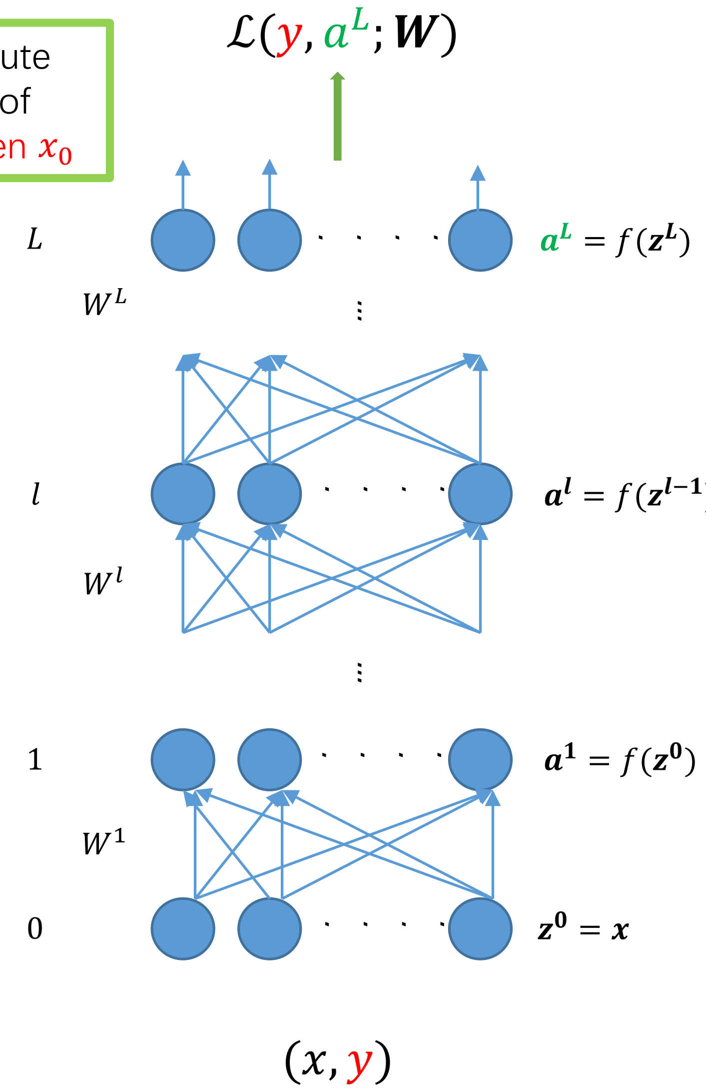
Gradient Computation

- $g_0 = f(x) = e^{\sin(x^2)}$

We want to compute the specific value of the derivative given x_0



$$\left.\frac{\partial f}{\partial x}\right|_{x=x_0} = e^{g_{1_{x_0}}} \cdot \cos(g_{2_{x_0}}) \cdot 2x_0$$



- This is called Computational Graph!
- And this procedure is called **Backpropagation**.

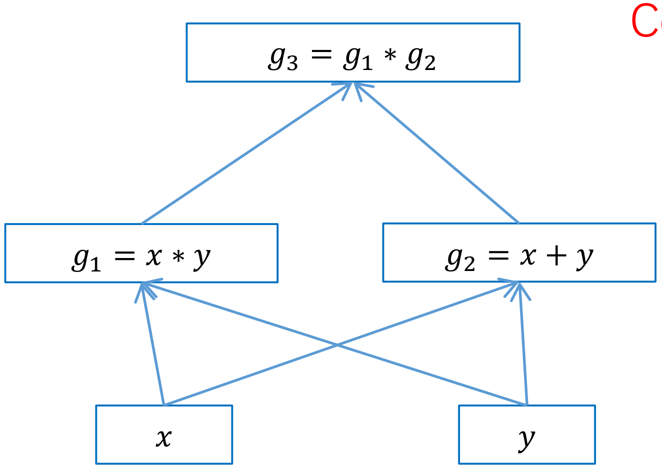
The Philosophy!

Back Propagation: A Nice Explain

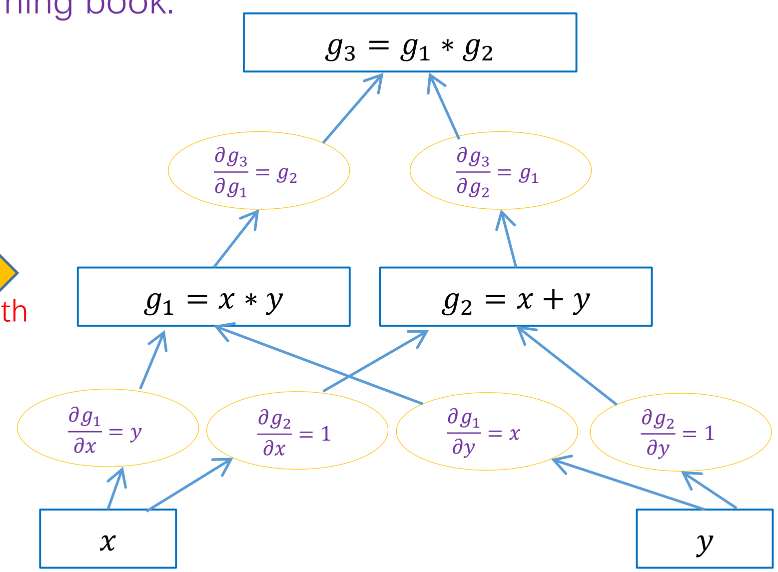
Computational Graph

- A Computational Graph is a way to depict function composition.
  - Node: specify elementary computation on in-edges
  - Alternative Node: store partial derivative along path
  - Directed Edge: specify computation flows and compositions.
- *"we use each node in the graph to indicate a variable. The variable may be a scalar, matrix, tensor [⋯] we also need to introduce the idea of an operation. An operation is a simple function of one or more variables. [⋯] If a variable y is computed by applying an operation to a variable x, then we draw a directed edge from x to y."* --- from Deep Learning book.

Compute: $\frac{\partial g_3}{\partial y}$?



Augment edge with form of derivative

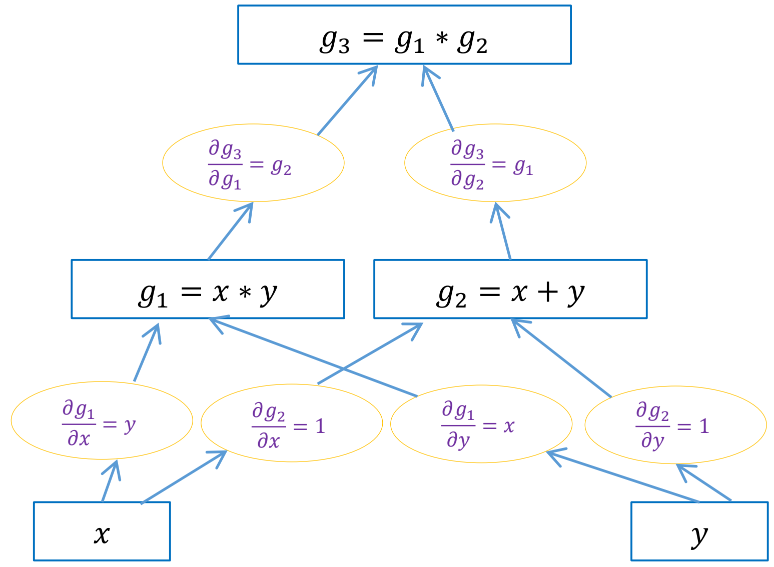


Back Propagation: A Nice Explain

Computational Graph

- A Computational Graph is a way to depict function composition.
  - Node: specify elementary computation on in-edges
  - Alternative Node: store partial derivative along path
  - Directed Edge: specify computation flows and compositions.

Compute: $\frac{\partial g_3}{\partial y}$?



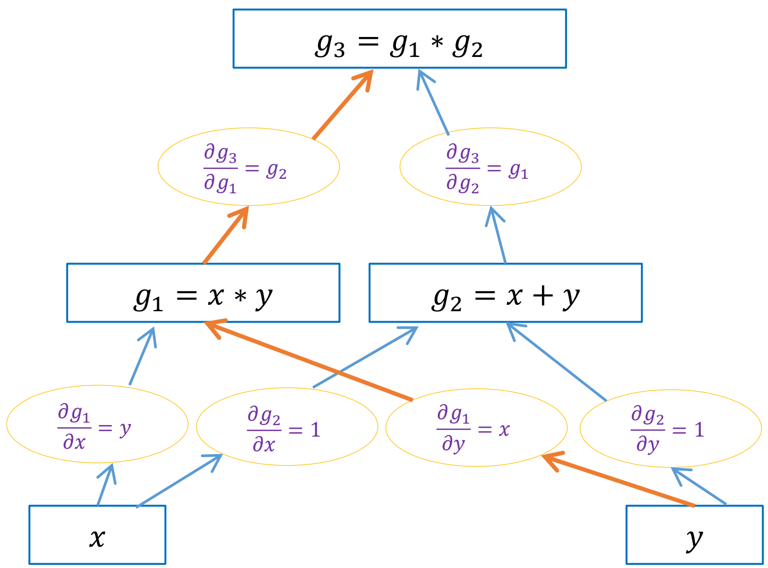
Back Propagation: A Nice Explain

Computational Graph

- A Computational Graph is a way to depict function composition.
  - Node: specify elementary computation on in-edges
  - Alternative Node: store partial derivative along path
  - Directed Edge: specify computation flows and compositions.

Compute: $\frac{\partial g_3}{\partial y}$?

$$\frac{\partial g_3}{\partial y} = \frac{\partial g_3}{\partial g_1} \cdot \frac{\partial g_1}{\partial y} +$$



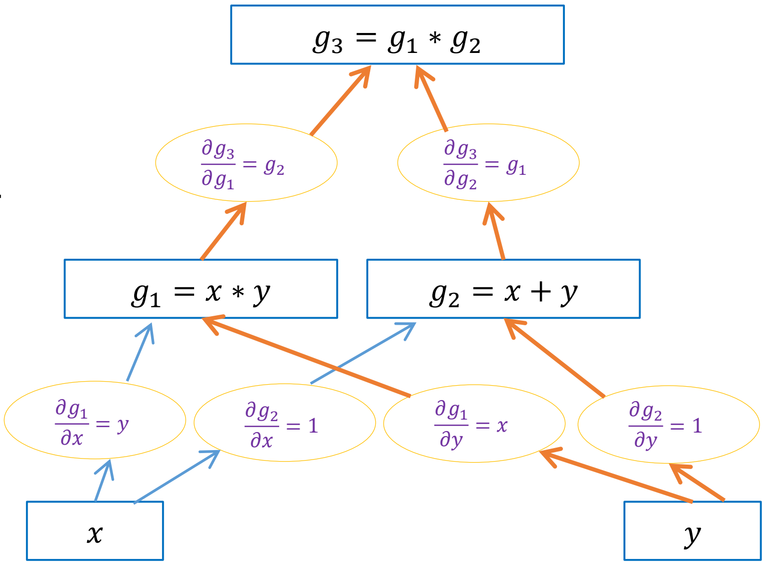
Back Propagation: A Nice Explain

Computational Graph

- A Computational Graph is a way to depict function composition.
  - Node: specify elementary computation on in-edges
  - Alternative Node: store partial derivative along path
  - Directed Edge: specify computation flows and compositions.

Compute: $\frac{\partial g_3}{\partial y}$?

$$\frac{\partial g_3}{\partial y} = \frac{\partial g_3}{\partial g_1} \cdot \frac{\partial g_1}{\partial y} + \frac{\partial g_3}{\partial g_2} \cdot \frac{\partial g_2}{\partial y}$$



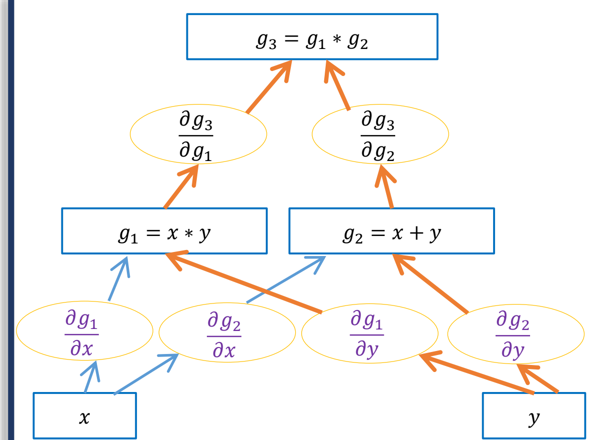
Back Propagation: A Nice Explain

Computational Graph

$$g_3 = f(g_1, g_2)$$

$$g_1 = p(x,y) \qquad g_2 = q(x,y)$$

$$\frac{\partial g_3}{\partial y} = \frac{\partial g_3}{\partial g_1} \cdot \frac{\partial g_1}{\partial y} + \frac{\partial g_3}{\partial g_2} \cdot \frac{\partial g_2}{\partial y}$$



$$\mathcal{L} = \mathcal{L}oss(\boldsymbol{a}^L)$$

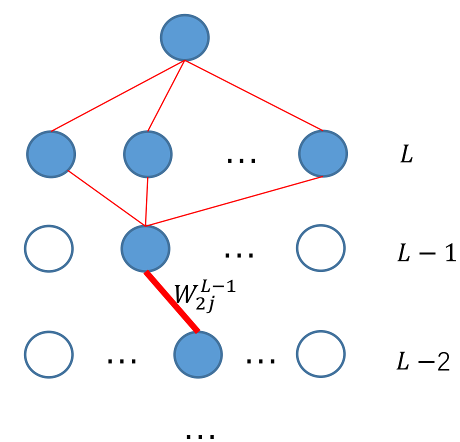
$$\boldsymbol{a}^L = f^L(\boldsymbol{z}^L)$$

$$\boldsymbol{z}^L = W^L \boldsymbol{a}^{L-1}$$

$$\boldsymbol{a}_2^{L-1} = f^{L-1}(\boldsymbol{z}_2^{L-1})$$

$$\boldsymbol{z}_2^{L-1} = W_2^{L-1} \boldsymbol{a}^{L-2}$$

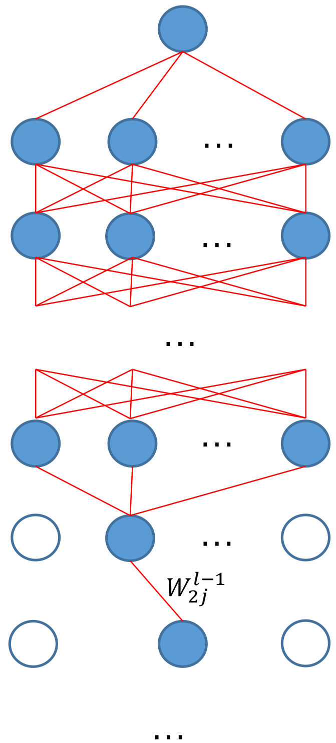
$$\frac{\partial \mathcal{L}}{\partial W_{2j}^{L-1}} = \sum_i \frac{\partial \mathcal{L}}{\partial a_i^L} \cdot \frac{\partial a_i^L}{\partial z_i^L} \cdot \frac{\partial a_i^L}{\partial z_i^L} \cdot \frac{\partial z_i^L}{\partial a_2^{L-1}} \cdot \frac{\partial a_2^{L-1}}{\partial z_2^{L-1}} \cdot \frac{\partial a_2^{L-1}}{\partial W_j^{L-1}}$$



Back Propagation: A Nice Explain

Partial Derivative Chain Rule

- What if we encounter:



The Partial Derivative Chain Rule:

If $p \in P$, where P is a set of paths from W_{ij}^{l} to the loss node $\mathcal{L}$, then

$$\frac{\partial \mathcal{L}}{\partial W_{ij}^{l}} = \sum_{p \in P} \prod_{e=(v_2 \rightarrow v_1) \in p} \frac{\partial v_1}{\partial v_2}$$

However, this still proposes a problem:

- Computation grows exponentially while the linear growth of layer number.

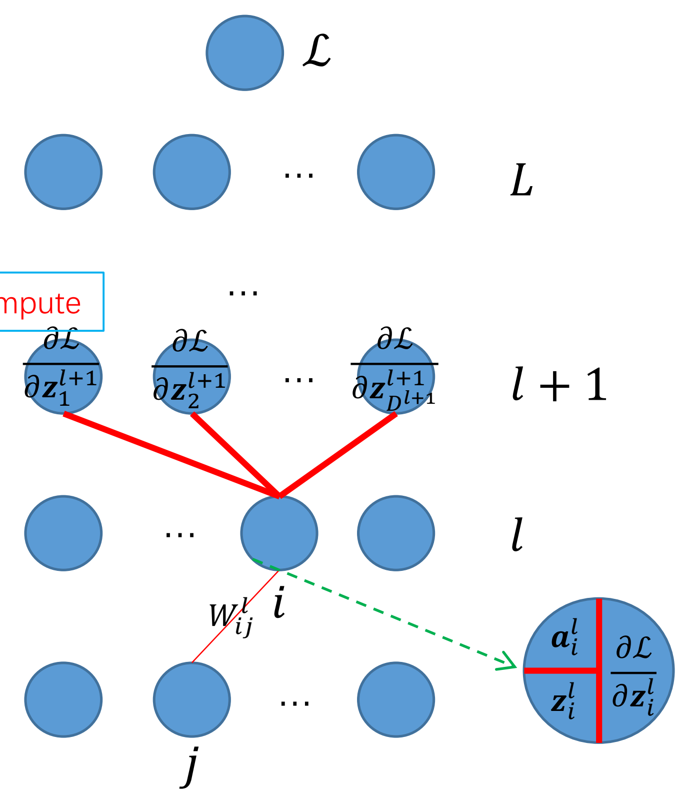
Can we use a method to reduce computation cost?

- ~~Divide and conquer?~~
- $\boxed{\text{Dynamic programming?}}$

Back Propagation: A Nice Explain

Backpropagation

- Let us consider the value of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_i^l}$ $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}\right)$
- Recall we are targeting on computing $\frac{\partial \mathcal{L}}{\partial W_{ij}^l}$,
  - which could be written as $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_i^l} \cdot \boxed{\frac{\partial z_i^l}{\partial W_{ij}^l}}$ Trivial to compute
- Suppose when we compute the l-th layer, we have $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l+1}}$, which is a D^{l+1} dimensional vector
  - let us see whether computing can be recursive.

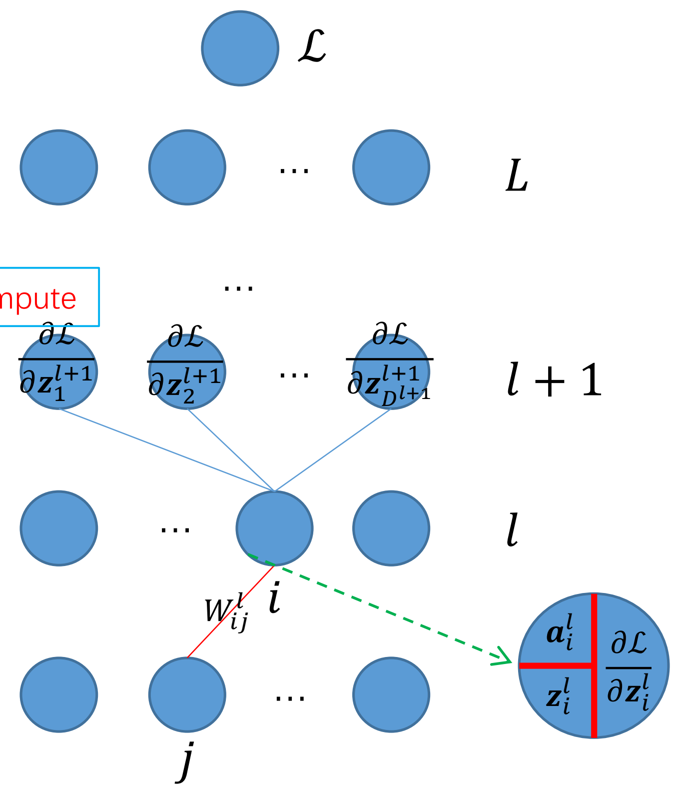


$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_i^l} = \boxed{\sum_{j=1}^{D^{l+1}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_j^{l+1}} \cdot \frac{\partial \boldsymbol{z}_j^{l+1}}{\partial \boldsymbol{a}_i^l}} \cdot \frac{d\boldsymbol{a}_i^l}{d\boldsymbol{z}_i^l}$$

Back Propagation: A Nice Explain

Backpropagation

- Let us consider the value of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_i^l}$ $(\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l})$
- Recall we are targeting on computing $\frac{\partial \mathcal{L}}{\partial W_{ij}^l}$,
  - which could be written as $\frac{\partial \mathcal{L}}{\partial z_i^l} \cdot \boxed{\frac{\partial z_i^l}{\partial W_{ij}^l}}$ Trivial to compute
- Suppose when we compute the l-th layer, we have $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l+1}}$, which is a D^{l+1} dimensional vector
  - let us see whether computing can be recursive.
- This eases the computation of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}$ by:
  - $\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l+1}} \cdot W^{l+1^T}) \odot \text{diag}(\nabla_{\boldsymbol{z}^l} \boldsymbol{a}^l)$



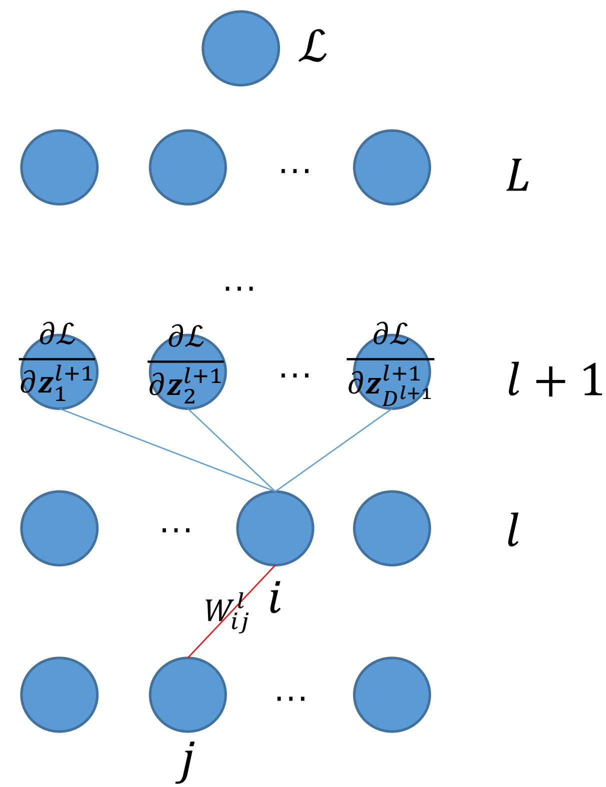
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}_i^l} = \boxed{\frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l+1}} \cdot W_{\cdot i}^{l+1}} \cdot \frac{d\boldsymbol{a}_i^l}{d\boldsymbol{z}_i^l}$$

Back Propagation: A Nice Explain

Backpropagation

Algorithm: The Backpropagation

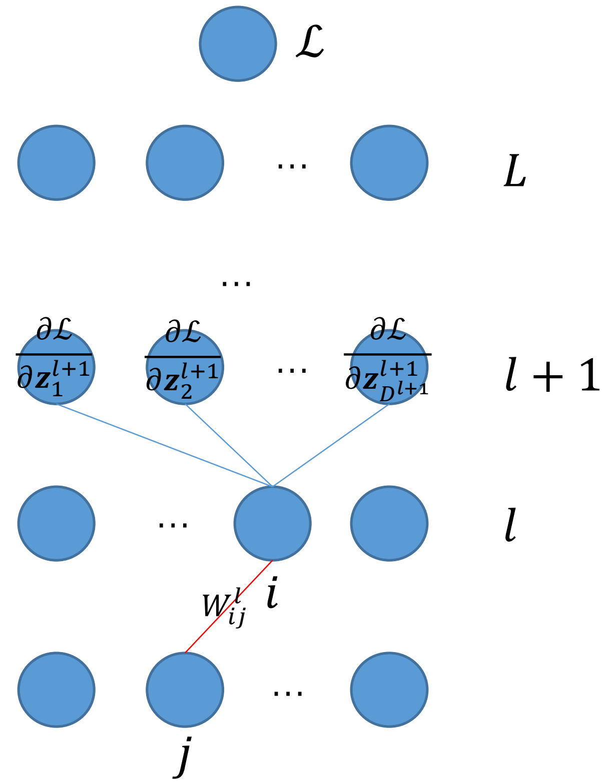
Input: A symbolic representation of a the function described by the network, Weights W^l for each layer
##Forward Computation##
If given the input $\boldsymbol{x}$
Set $\boldsymbol{z}^0 = \boldsymbol{x}$
For $l = 1$ to L
  Compute $z^l = W^l z^{l-1}$, $a^l = f(z^l)$
End For
##Backward Computation ##
For $l = L$ **to** 1
  If $l = L$
    Compute $\eth^L = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^L}$ and store in each unit of layer L
  Else
    Compute $\eth^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l} = (\eth^{l+1} \cdot W^{l+1^T}) \odot diag(\nabla_{\boldsymbol{z}^l} \boldsymbol{a}^l)$
    And compute $\frac{\partial \mathcal{L}}{\partial W^l_{ij}} = \eth^l_i \cdot \boldsymbol{a}^l_j$
End For



Back Propagation: A Nice Explain

Matrix Form

- Theorem:
  - For a vector function $f: \mathbb{R}^{V^l} \to \mathbb{R}^{V^{l+1}}$, that is $a^{l+1} = f(a^l)$, we have the Jacobian matrix of f w.r.t. a^l, as:
    - $Jacob\left(\frac{\partial a^{l+1}}{\partial a^l}\right) = \begin{bmatrix} \frac{\partial a_1^{l+1}}{\partial a_1^l} & \cdots & \frac{\partial a_1^{l+1}}{\partial a_{V^l}^l} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{V^{l+1}}^{l+1}}{\partial a_1^l} & \cdots & \frac{\partial a_{V^{l+1}}^{l+1}}{\partial a_{V^l}^l} \end{bmatrix}$
  - If vector function $f^{l+1}: \mathbb{R}^{V^l} \to \mathbb{R}^{V^{l+1}}$, that is $a^{l+1} = f(a^l)$; and we also have $f^{l+2}: \mathbb{R}^{V^{l+1}} \to \mathbb{R}^{V^{l+2}}$, that is $a^{l+2} = f(a^{l+1})$, the Jacobian matrix of f^{l+2} w.r.t. a^l is:
    - $Jacob\left(\frac{\partial a^{l+2}}{\partial a^{l+1}}\right) \cdot Jacob\left(\frac{\partial a^{l+1}}{\partial a^l}\right)$



Outline

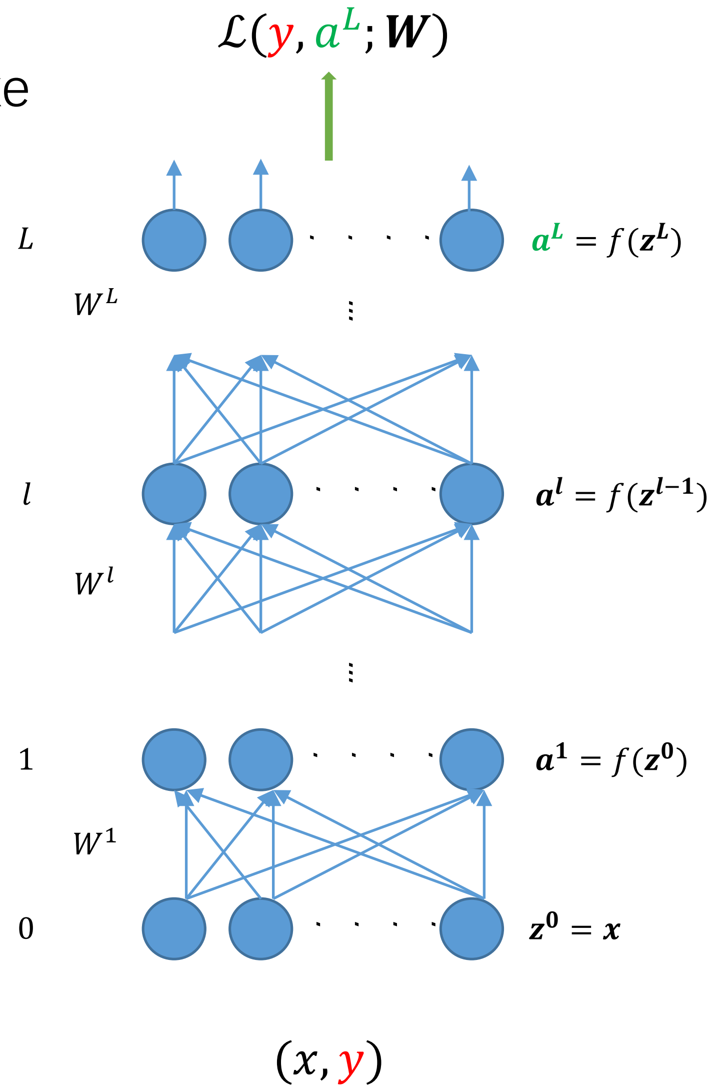
- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Regularization

What and Why?

- Regularization: is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
  - E.g. Weight decay. $-logp(y|\boldsymbol{x}, W) + ||W||_2$

- The concept of **capacity!**

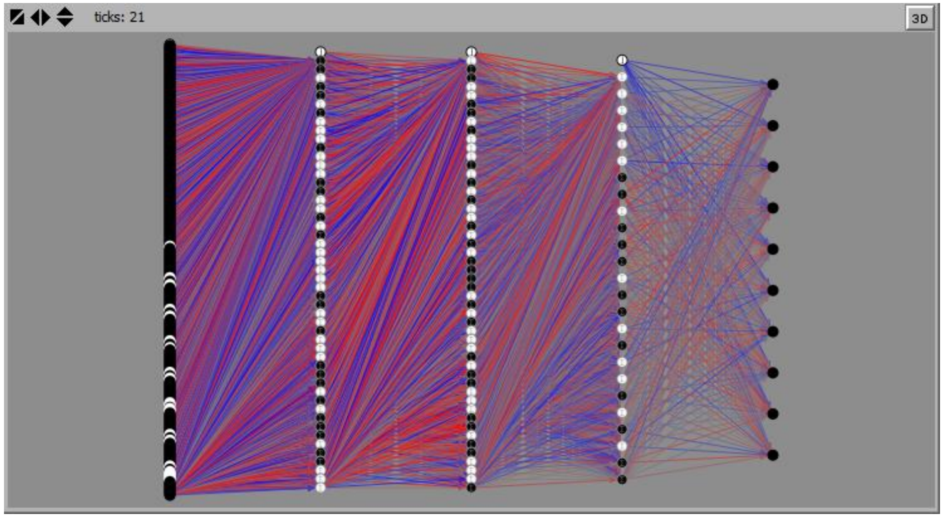


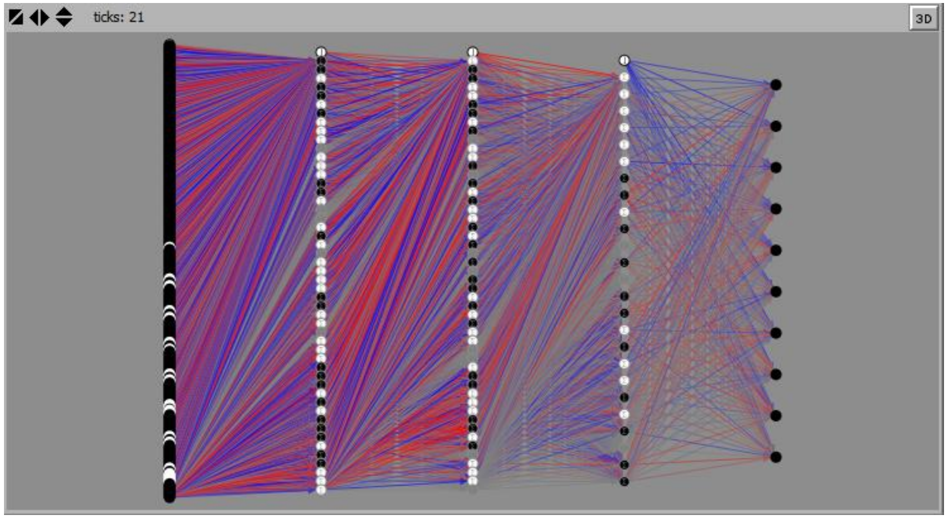
Outline

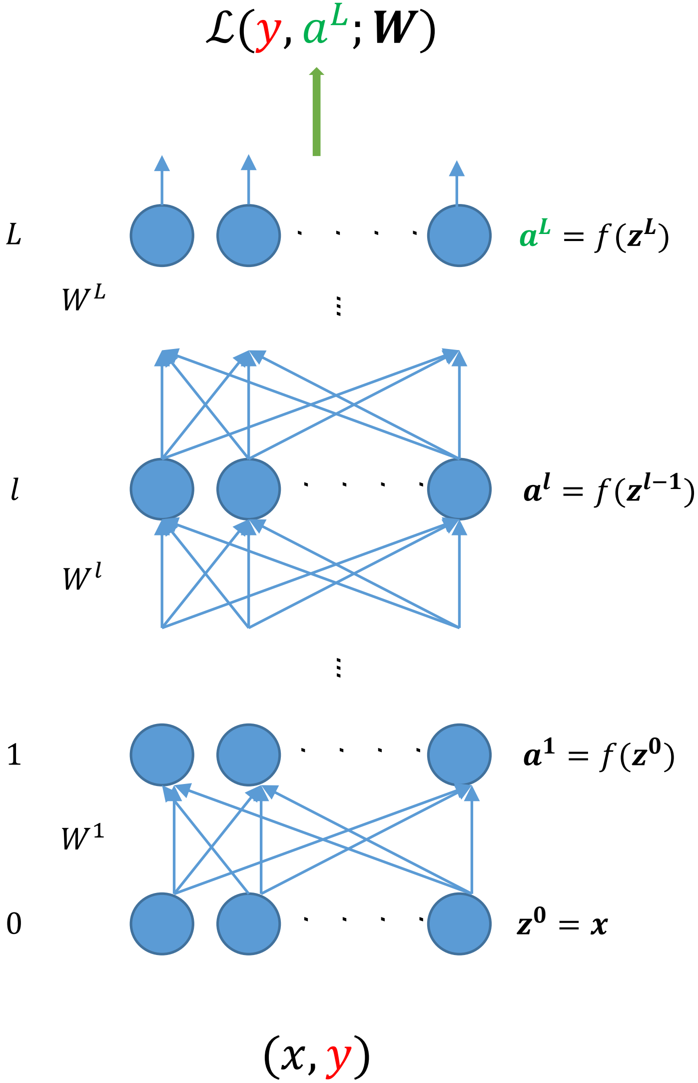
- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - **Dropout**
- Optimization Tips
- Distilling the Book

Regularization

Dropout







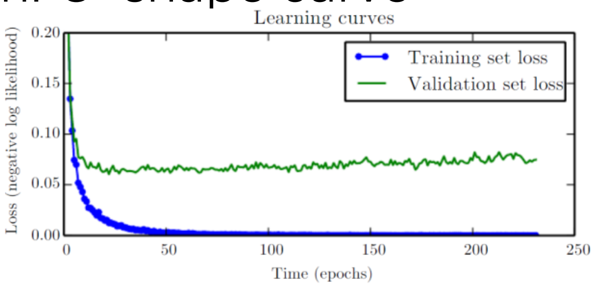
Outline

- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - **Early stop**
  - Dropout
- Optimization Tips
- Distilling the Book

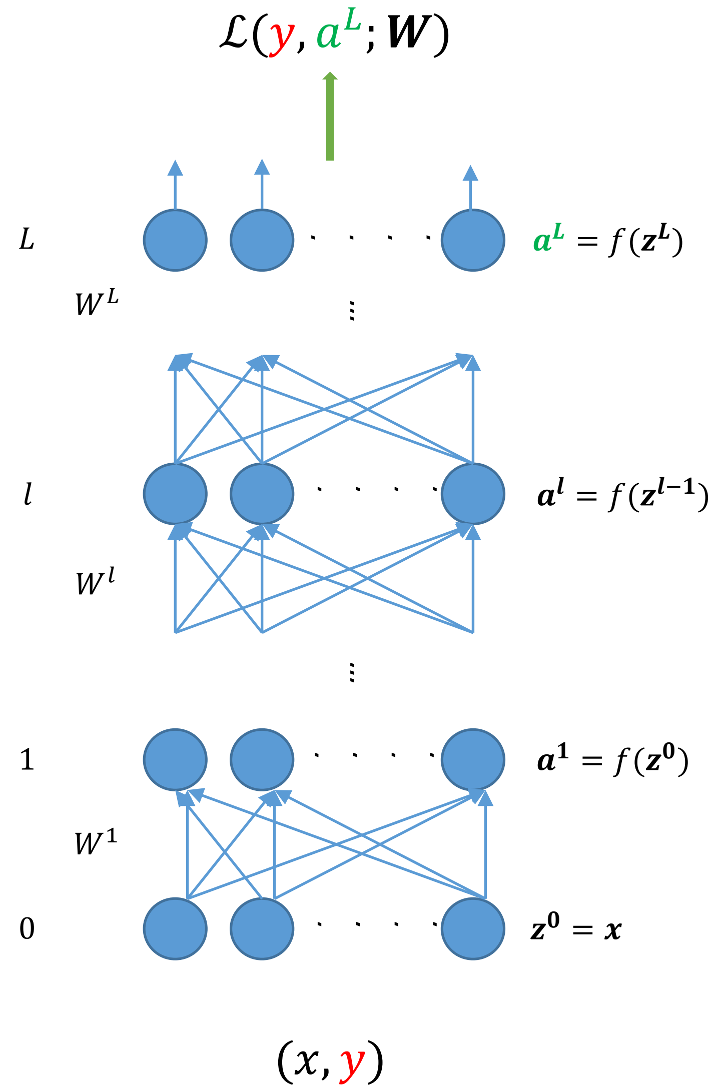
Regularization

Early Stopping

- Intuition: U-shape curve



- The idea of early stopping is very simple and heuristic.
  - We split our dataset into training and validation set.
  - While doing gradient descent on training set, we keep an eye on the validation set error.
  - We choose to have a patience: how many times the validation set error start to rise instead of steadily falling down.
  - Each time of validation set error rising we store parameters value at that time.
  - We stop while reach our patience.



Outline

- Warming Up
- Building Blocks of A Neural Network
  - Architecture
  - Functionality
  - Design
- Back Propagation: A Nice Explanation
  - Matrix Form, Computational Graph
- Regularization
  - Early stop
  - Dropout
- Optimization Tips
- Distilling the Book

Optimization Tips

Stochastic Gradient Descent

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k
Require: Initial parameter $\boldsymbol{\theta}$
While stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ with corresponding $y^{(i)}$s
Compute gradient estimate:
$\hat{\boldsymbol{g}} \leftarrow + \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i \mathcal{L}(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)})$
Apply update:
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
End While

- Why stochastic? Not on the whole batch? Why call it gradient estimate?
  - Computation tradeoff
- We could prove that with Large Number Theory
  - The expected value of gradient from stochastic minibatch sample equals the averaged real gradient over the whole batch

Optimization Tips

Stochastic Gradient Descent

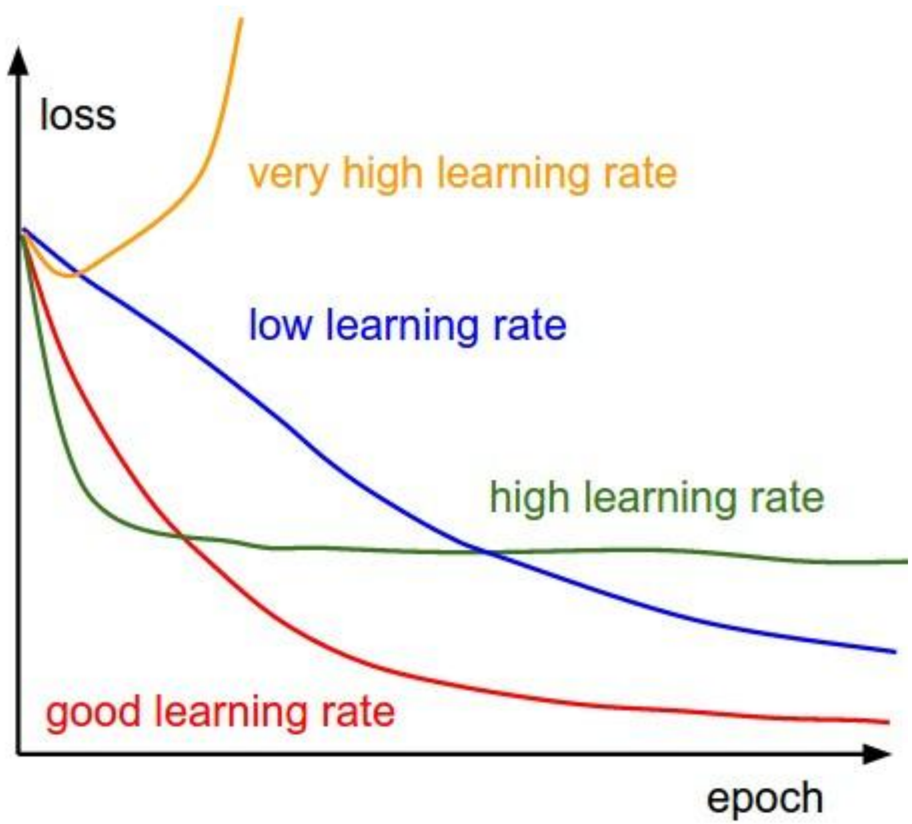
- In every optimization algorithm the hyperparameters are under our consideration of selection.

- The minibatch size:

  .erion not met c
  ch of m example
  t estimate:

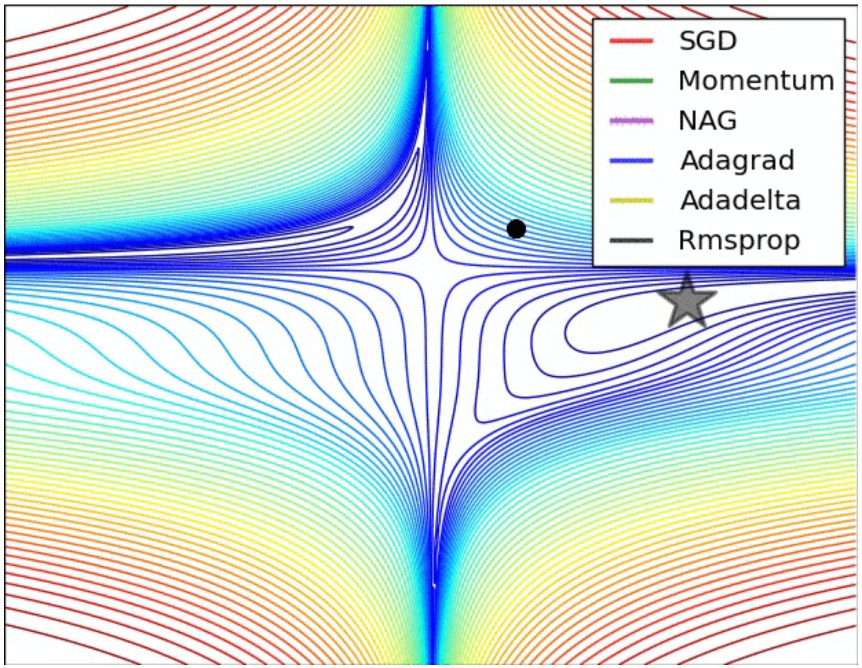
- And the learning rate:

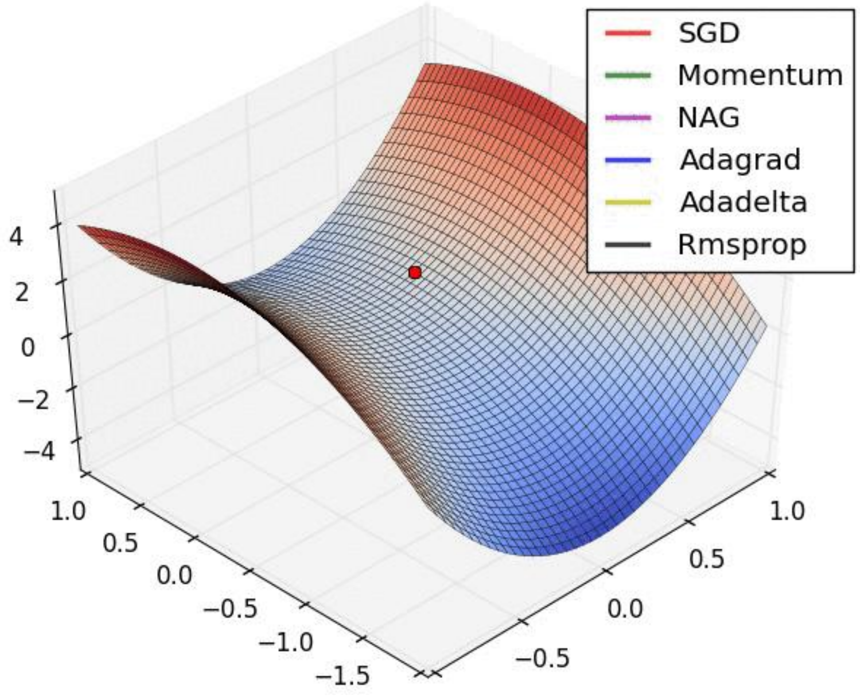
  : Learning rate ϵ_k
  : Initial parameter $\boldsymbol{\theta}$



Optimization Tips

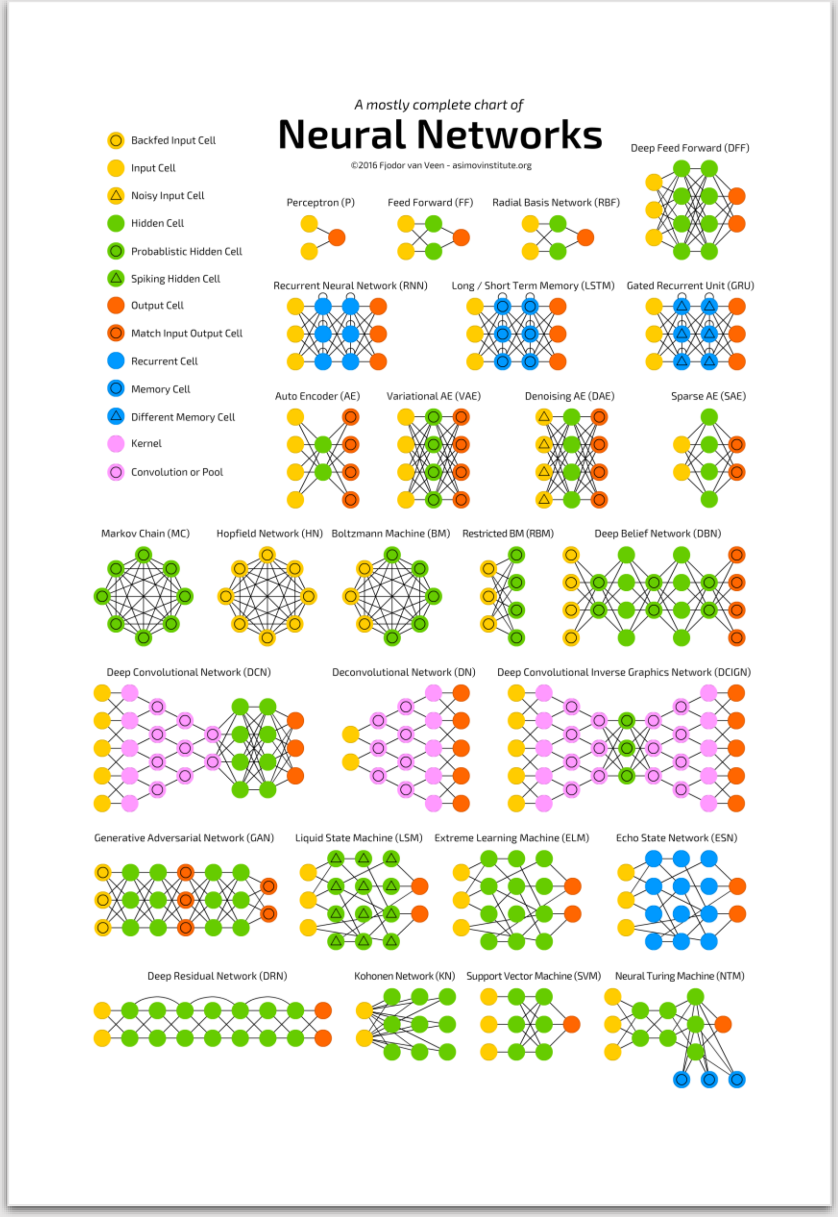
Adaptive Learning Rate



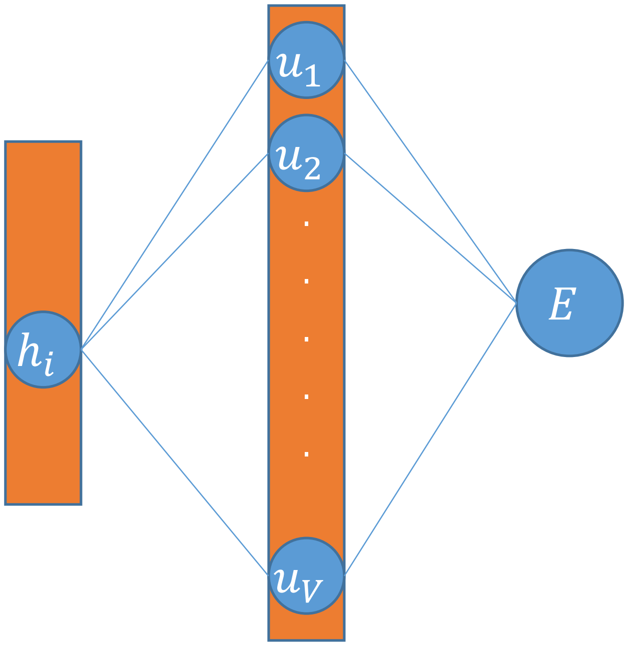


Bonus!

- Zoo of Neural Networks



Nothing here.



$$E = u_1 * u_2$$
$$u_1 = x + y$$
$$u_2 = x - y$$

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial u_1}\frac{\partial u_1}{\partial x} + \frac{\partial E}{\partial u_2}\frac{\partial u_2}{\partial x}$$

$$\Downarrow$$

$$E = f(g_1, \dots, g_m)$$
$$g_i = f_i(x, y)$$

$$\frac{\partial E}{\partial x} = \sum_i \frac{\partial E}{\partial g_i}\frac{\partial g_i}{\partial x}$$