Basic Machine Learning
Supervised Learning

Kyunghyun Cho
New York University
Courant Institute (Computer Science) and Center for Data Science
Facebook AI Research

Data-Driven Algorithm Design?

- Algorithm: a sequence of instructions that solves a problem.
- Traditionally,
    1. A detailed problem specification is given.
    2. An algorithm is designed to solve the described problem.
- Machine learning: data-driven algorithm design
    1. [A rough problem specification is given.] ← optional
    2. A set of examples is provided.*
    3. A machine learning model is "trained" to solve the problem.
- A meta-approach to algorithm design

* Examples could be provided in a variety of ways.

Supervised Learning – Overview

- Provided:
  1. a set of N input-output “training” examples
     $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  2. A per-example loss function*
     $l(M(x), y) \geq 0$
  3. Evaluation sets*: validation and test examples $D_{\text{val}}, D_{\text{test}}$
- What we must decide:
  1. Hypothesis sets $\mathcal{H}_1, \ldots, \mathcal{H}_M$
     - Each set consists of all compatible models
  2. Optimization algorithm

* Often it is necessary to design a loss function.

* Often these sets are created by holding out subsets of training examples.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  1. For each hypothesis set $\mathcal{H}_m$, find the best model:*

$$\hat{M}_m = \arg \min_{M \in \mathcal{H}_m} \sum_{n=1}^{N} l(M(x_n), y_n)$$

using the optimization algorithm.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  1. [Training] For each hypothesis set $\mathcal{H}_m$, find the best model:*

$$\hat{M}_m = \arg \min_{M \in \mathcal{H}_m} \sum_{n=1}^{N} l(M(x_n), y_n)$$

     using the optimization algorithm and the **training set**.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  2. [Model Selection]* Among the trained models, select the best one

$$\hat{M} = \arg\min_{M \in \{\mathcal{H}_1, \ldots, \mathcal{H}_M\}} \sum_{(x,y) \in D_{\text{val}}} l(M(x), y)$$

using the **validation set** loss.

* If you're familiar with deep learning, "hyperparameter optimization" may be a more familiar term for you.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  3. [Reporting] Report how well the best model *would* work

$$R(\hat{M}) \approx \frac{1}{|D_{\text{test}}|} \sum_{(x,y) \in D_{\text{test}}} l(\hat{M}(x), y)$$

  using the **test set** loss.

* If you're familiar with deep learning, "hyperparameter optimization" may be a more familiar term for you.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
- It results in an algorithm $\hat{M}$ with an expected performance of $R(\hat{M})$.

Supervised Learning

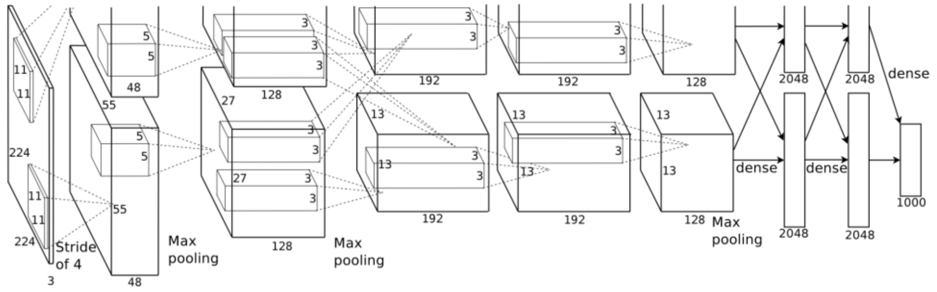
- Three points to consider both in research and in practice
    1. How do we decide/design a **hypothesis set**?
    2. How do we decide a **loss function**?
    3. How do we **optimize** the loss function?

Hypothesis set – Neural Networks

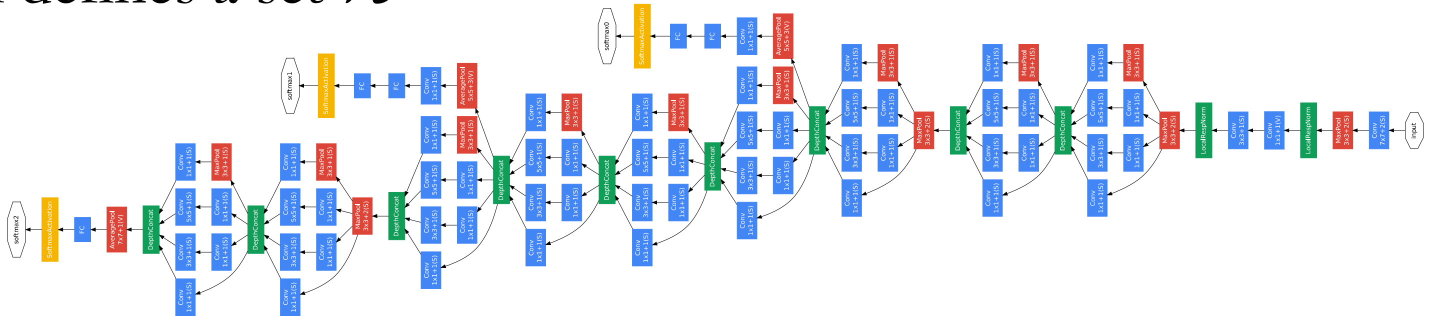
- What kind of machine learning approach will we consider?
  - Classification:
    - Support vector machines, Naïve Bayes classifier, logistic regression, …?
  - Regression:
    - Support vector regression, Linear regression, Gaussian process, …?
- How are the hyperparameters sets?
  - Support vector machines: regularization coefficient C
  - Gaussian process: kernel function $k(\cdot, \cdot)$

Hypothesis set – Neural Networks

- In the case of deep learning/artificial neural networks,
  1. The architecture of a network defines a set $\mathcal{H}$



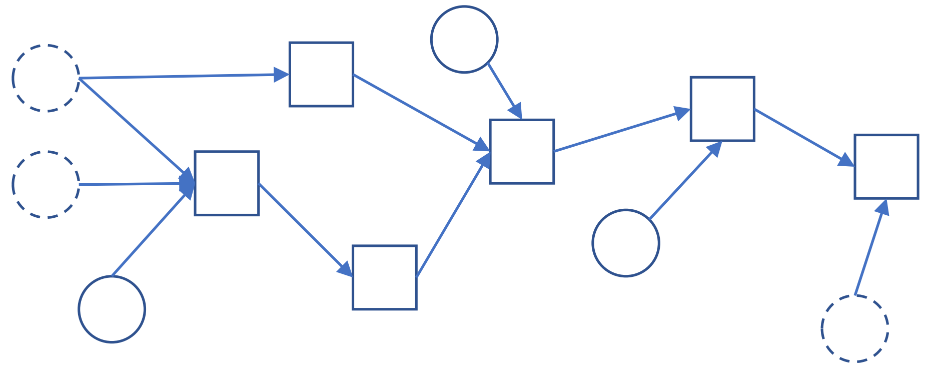
vs.



  2. Each model in the set $M \in \mathcal{H}$ is characterized by its parameters θ
     - Weights and bias vectors define one model in the hypothesis set.
- There are infinitely many models in a hypothesis set.
- We use optimization to find "a" good model from the hypothesis set.

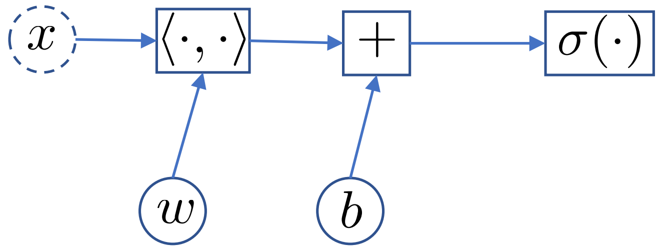
Network Architectures

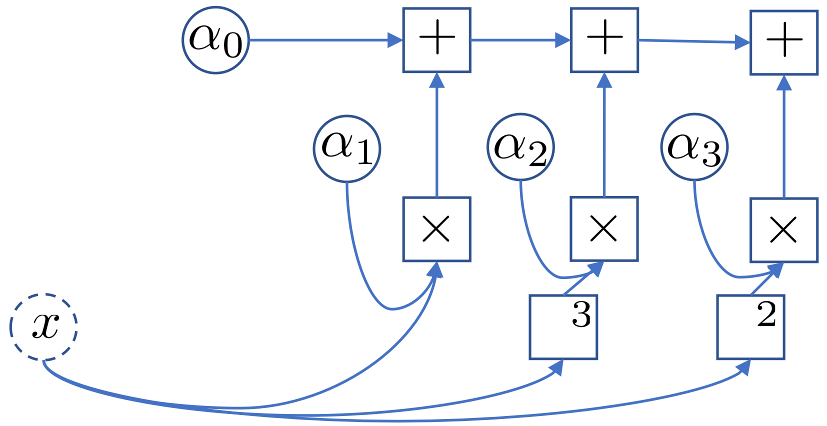
- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)



1. Solid Circles ○ : parameters (to be estimated or found)
2. Dashed Circles ◌ : vector inputs/outputs (given as a training example)
3. Squares □ : compute nodes (functions, often continuous/differentiable)

Network Architectures

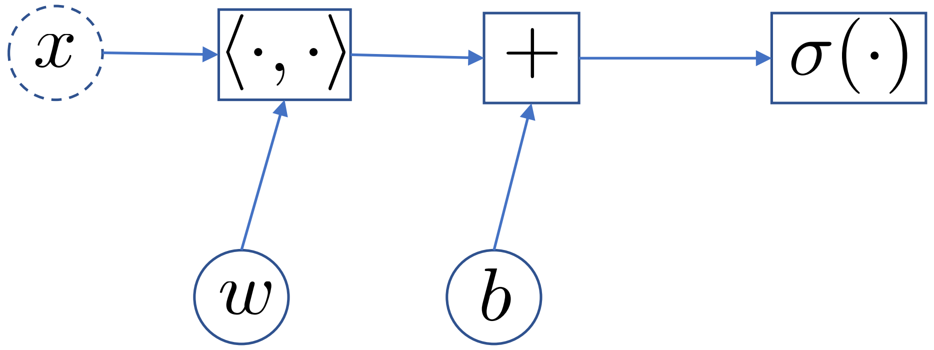
- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
    1. Logistic regression
    $$p_\theta(y=1|x)=\sigma(w^\top x+b)=\frac{1}{1+\exp(-w^\top x-b)}$$
    

    2. 3rd-order polynomial function $y=\alpha_0+\alpha_1x+\alpha_2x^2+\alpha_3x^3$
    

Inference – Forward Computation

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Forward computation: how you "use" a trained neural network.
  - Topological sweep (breadth-first)
  - Logistic regression

$$p_\theta(y=1|x) = \sigma(w^\top x + b) = \frac{1}{1+\exp(-w^\top x - b)}$$



DAG ↔ Hypothesis Set

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Implication in practice
  - Naturally supports high-level abstraction
  - Object-oriented paradigm fits well.*
    - Base classes: variable (input/output) node, operation node
    - Define the internal various types of variables and operations by inheritance
  - Maximal code reusability
    - See the success of PyTorch, TensorFlow, DyNet, Theano, …
- You define a hypothesis set by designing a directed acyclic graph.
- The hypothesis space is then a set of all possible parameter settings.

* Functional programming as well ☺

Supervised Learning

- Three points to consider both in research and in practice
    1. How do we decide/design a **hypothesis set**?
    2. How do we decide a **loss function**?
    3. How do we **optimize** the loss function?

Loss Functions

- Per-example loss function
  - Computes how good a model is doing on a given example:
    $l(M(x), y) \geq 0$
- So many loss functions…
  - Classification: hinge loss, log-loss, …
  - Regression: mean squared error, mean absolute error, robust loss, …
- In this lecture, we stick to distribution-based loss functions.

Probability in 5 minutes – (1)

- An "event set" Ω contains all possible events: $\Omega = \{e_1, e_2, \ldots, e_D\}$
  - Discrete: when there are a finite number of events $|\Omega| < \infty$
  - Continuous: when there are infinitely many events $|\Omega| = \infty$
- A "random variable" X could take any one of these events: $X \in \Omega$
- A probability of an event: $p(X = e_i)$
  - How likely would the *i*-th event happen?
  - How often has the *i*-th event occur relative to the other events?
- Properties
  1. Non-negative: $p(X = e_i) \geq 0$
  2. Unit volume: $\sum_{e \in \Omega} p(X = e) = 1$

Probability in 5 minutes – (2)

- Multiple random variables: consider two here - X, Y
- A joint probability $p(Y = e_j^Y, X = e_i^X)$
  - How likely would e_j^Y and e_i^X happen together?
- A conditional probability $p(Y = e_j^Y | X = e_i^X)$
  - Given e_i^X, how likely would e_j^Y happen?
  - The chance of both happening together divided by that of e_i^X happening regardless of whether e_j^Y happened:

$$p(Y|X) = \frac{p(X,Y)}{p(X)} \iff p(X,Y) = p(Y|X)p(X)$$

- Probability function $p(X)$ returns a probability of X

Probability in 5 minutes – (3)

- Multiple random variables: consider two here - X, Y
- A joint probability $p(Y = e_j^Y, X = e_i^X)$
  - How likely would e_j^Y and e_i^X happen together?
- A marginal probability $p(Y = e_j^Y)$
  - Regardless of what happens to X, how likely is e_j^Y?

$$p(Y = e_j^Y) = \sum_{e \in \Omega_X} p(Y = e_j^Y, X = e)$$

A Neural network computes a conditional distribution

- Supervised learning: what is *y* given *x*?

$$f_\theta(x) = ?$$

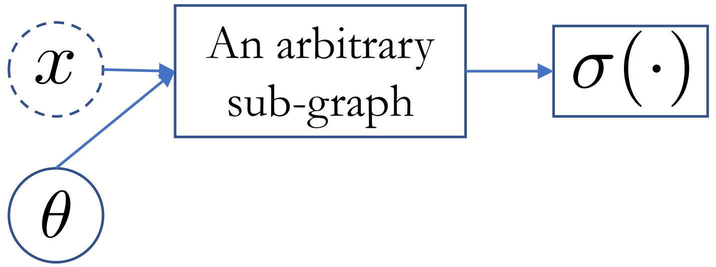
- In other words, how probable is a certain value *y'* of *y* given *x*?

$$p(y = y'|x) = ?$$

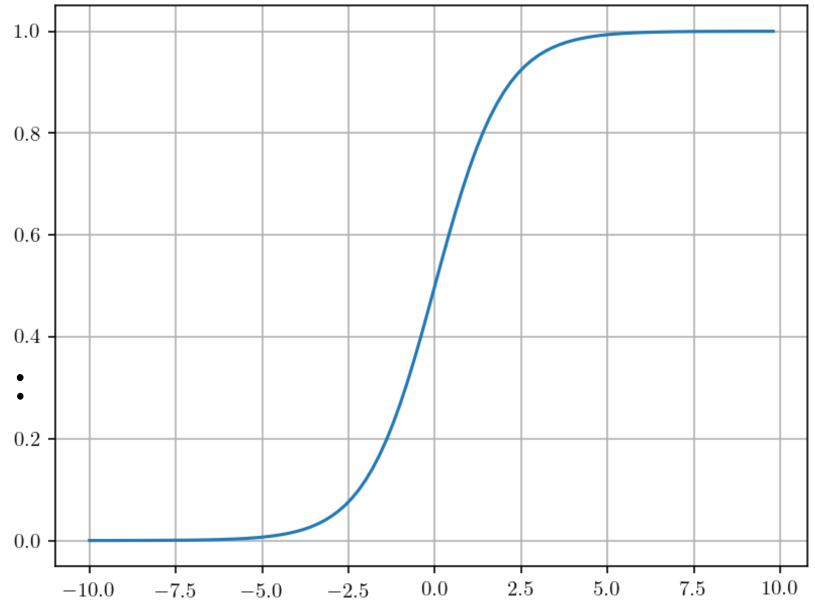
- What kind of distributions?
  - Binary classification: Bernoulli distribution
  - Multiclass classification: Categorical distribution
  - Linear regression: Gaussian distribution
  - Multimodal linear regression: Mixture of Gaussians

Important distributions – Bernoulli

- How probable is a certain value y' of y given x? $p(y = y'|x) = ?$
- Binary classification: Bernoulli distribution $\mathcal{B}(\mu)$
  - Probability: $p(y|x) = \mu^y (1-\mu)^{1-y}$, where $y \in \{0, 1\}$
  - Fully characterized by $\mu \in [0, 1]$.
  - A neural network then should turn the input x into μ

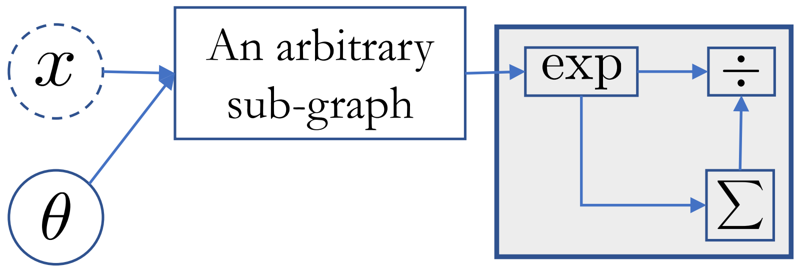


using a sigmoid function $\sigma(a) = \dfrac{1}{1+\exp(-a)}$:



Important distributions – Categorical

- How probable is a certain value y' of y given x? $p(y = y'|x) =?$
- Multi-class classification: Categorical distribution $\mathcal{C}(\{\mu_1, \mu_2, \ldots, \mu_C\})$
  - Probability: $p(y = v|x) = \mu_v$, where $\sum_{v=1}^{C} \mu_v = 1$
  - Fully characterized by $\{\mu_1, \mu_2, \ldots, \mu_C\}$.
  - A neural network then should turn the input x into a vector $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_C \end{bmatrix}$



using a **softmax** function: $\mathrm{softmax}(a) = \frac{1}{\sum_{v=1}^{C} \exp(a_v)} \exp(a)$.

Important distributions – Gaussian

- How probable is a certain value y' of y given x? $p(y = y'|x) =?$
- Regression: Gaussian distribution $\mathcal{N}(\mu, \mathbb{I})$ with an identity covariance
  - Probability: $p(y|x) = \frac{1}{Z}\exp(-\frac{1}{2}(y-\mu)^\top(y-\mu))$
  - Fully characterized by $\mu \in \mathbb{R}^q$.
  - A neural network then should turn the input x into a vector μ.
  - Can be done trivially by affine transformation.

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_\theta(y|x)$, a natural way to define a loss function arises.
- Make sure training data is maximally likely:
  - Equiv. to making sure each and every training example is maximally likely.

$$\arg\max_\theta \log p_\theta(D) = \arg\max_\theta \sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

  - Why *log*? – many reasons… but out of the lecture's scope.
- Equivalently, we want to minimize the *negative* log-probability.
  - A loss function is the sum of negative log-probabilities of correct answers.

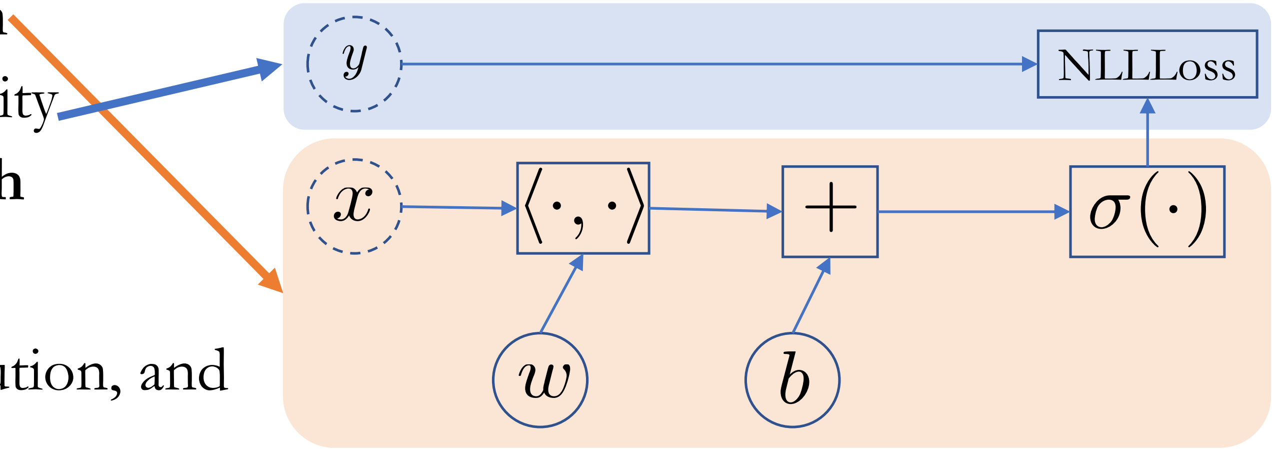
$$L(\theta) = \sum_{n=1}^{N} l(M_\theta(x_n), y_n) = -\sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_\theta(y|x)$, a natural way to define a loss function arises.
- Practical implications
  - An OP node: negative log-probability (e.g., NLLLoss in PyTorch)
    - Inputs: the conditional distribution and the correct output
    - Output: the negative log-probability (a scalar)

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_\theta(y|x)$, a natural way to define a loss function arises.
- Logistic regression
  - Computes a Bernoulli distribution
  - Computes a negative log-probability
  - All in **one directed acyclic graph**
- Forward computation
  - Computes the conditional distribution, and
  - Computes the per-example loss



Supervised Learning

- Three points to consider both in research and in practice
  1. How do we decide/design a **hypothesis set**?
  2. How do we decide a **loss function**?
  3. How do we **optimize** the loss function?

Loss Minimization

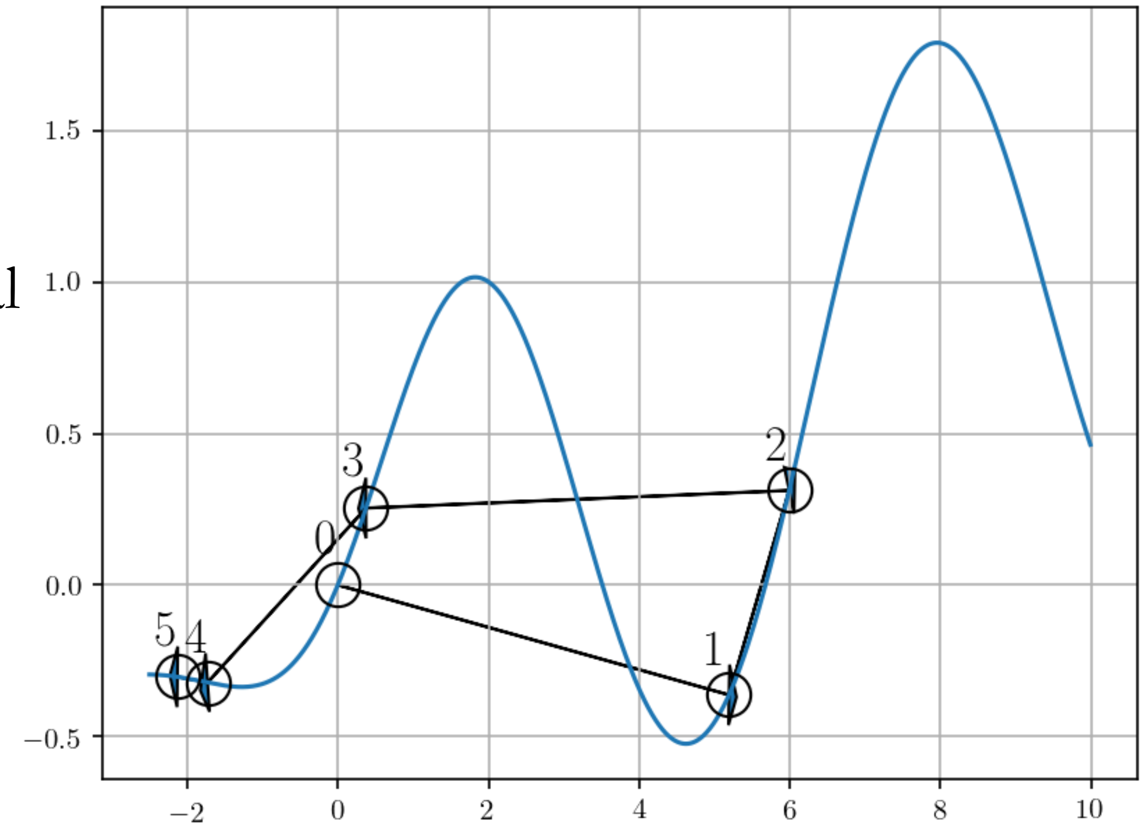
- What we now know
  1. How to build a neural network with an arbitrary architecture.
  2. How to define a per-example loss as a negative log-probability.
  3. Define a single directed acyclic graph containing both.
- What we now need to know
  1. Choose an optimization algorithm.
  2. How to use the optimization algorithm to estimate parameters θ.

Local, iterative optimization

- An arbitrary function $L : \mathbb{R}^d \to \mathbb{R}$
- Given the current value θ_0, how should I move to minimize L?
- Random guided search
  - Stochastically perturb θ_0: $\theta_k = \theta + \epsilon_k$, where $\epsilon_k \sim \mathcal{N}(0, s^2\mathbf{1})$
  - Test each perturbed point $L(\theta_k)$
  - Find the best perturbed point $\theta_1 = \arg\min_{\theta_k} L(\theta_k)$
  - Repeat this until no improvement could be made.
- Applicable to any arbitrary loss function and neural network.
- Inefficient in the high-dimensional parameter space: large *d*.

Local, iterative optimization

- An arbitrary function $L : \mathbb{R}^d \to \mathbb{R}$
- Given the current value θ_0, how should I move to minimize L?
- Random guided search
  - Applicable to any arbitrary loss function and neural network.
  - Inefficient in the high-dimensional parameter space



Gradient-based optimization

- A **continuous**, **differentiable*** function $L : \mathbb{R}^d \to \mathbb{R}$
- Given the current value θ_0, how should I move to minimize L?
- Gradient descent
  - The negative gradient of the function: $-\nabla L(\theta_0)$
  - This is only valid in a local neighbourhood of θ_0: take a very small step!

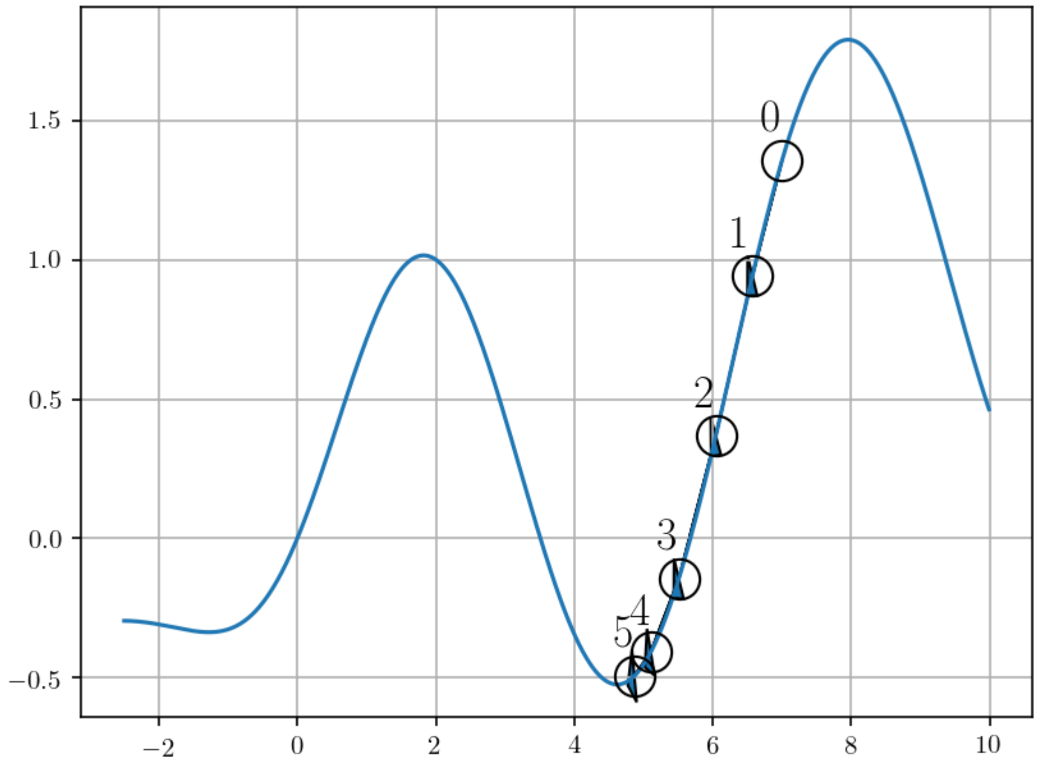
$$\theta = \theta_0 - \eta \nabla L(\theta_0)$$

- Efficient and effective even in the high dimensional space.
  - Can be improved with the second-order information (Hessian and/or FIM)

* Almost everywhere, but not necessarily everywhere

Gradient-based optimization

- A **continuous, differentiable** function $L : \mathbb{R}^d \to \mathbb{R}$
- Given the current value θ_0, how should I move to minimize L?
- Gradient descent
  - Efficient and effective even in the high dimensional space.
  - Learning rate must be carefully selected and annealed over time.



Backward Computation – Backpropagation

- How do we compute the gradient of the loss function?

1. Manual derivation
   - Relatively doable when the DAG is small and simple.
   - When the DAG is larger and complicated, too much hassle.
2. Automatic differentiation (autograd)
   - Use the chain rule of derivatives

$$\frac{\partial(f \circ g)}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

   - The DAG is nothing but a composition of (mostly) differentiable functions.
   - Automatically apply the chain rule of derivatives.

Backward Computation – Backpropagation

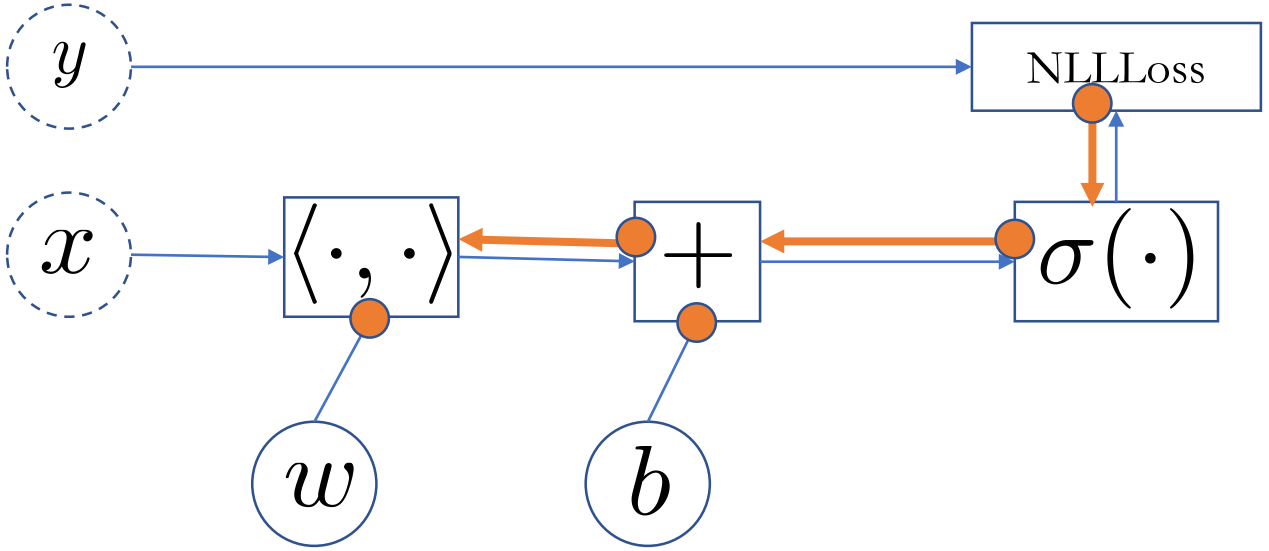
- Automatic differentiation (autograd)
  1. Implement the Jacobian-vector product of each OP node:

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \vdots \\ \frac{\partial L}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_{d'}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_1}{\partial x_d} & \cdots & \frac{\partial F_{d'}}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial F_1} \\ \vdots \\ \frac{\partial L}{\partial F_{d'}} \end{bmatrix}$$

    - Can be implemented efficiently without explicitly computing the Jacobian.
    - The same implementation can be reused every time the OP node is called.

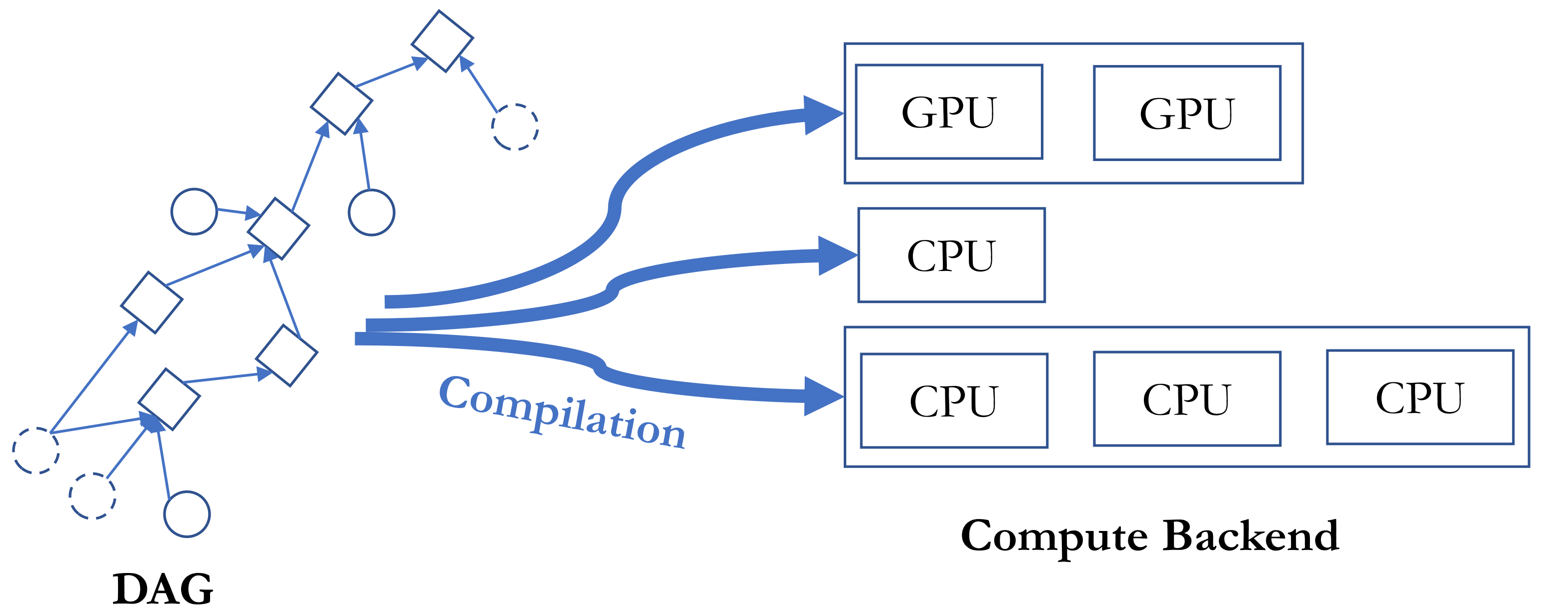
Backward Computation – Backpropagation

- Automatic differentiation (autograd)
  2. Reverse-sweep the DAG starting from the loss function node.
     - Iteratively multiplies the Jacobian of each OP node until the leaf nodes of the parameters.
     - As expensive as forward computation with a constant overhead: O(N), where N: # of nodes.



Backward Computation – Backpropagation

- Practical Implications – Automatic differentiation (autograd)
  - Unless a complete new OP is introduced, no need to manually derive the gradient
  - Nice de-coupling of specification (front-end) and implementation (back-end)
    1. [Front-end] Design a neural network by creating a DAG.
    2. [Back-end] The DAG is "compiled" into an efficient code for a target compute device.



Gradient-based Optimization

- Backpropagation gives us the gradient of the loss function w.r.t. θ
- Readily used by off-the-shelf gradient-based optimizers
  - Gradient descent, L-BFGS, Conjugate gradient, …
  - Though, most are not applicable in a realistic neural network with 10s or 100s of millions of parameters.
- Stochastic gradient descent
  - Approximate the full loss function (the sum of per-examples losses) using only a small random subset of training examples:

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

Stochastic Gradient Descent

- Stochastic gradient descent
  - Approximate the full loss function (the sum of per-examples losses) using only a small random subset of training examples:

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

  - Unbiased estimate of the full gradient.*
  - Learning rate must be annealed appropriately.
  - Extremely efficient *de facto* standard practice.

* Under certain conditions

Stochastic Gradient Descent

- Stochastic gradient descent in practice
  1. Grab a random subset of M training examples*
     $D' = \{(x_1, y_1), \ldots, (x_{N'}, y_{N'})\}$
  2. Compute the minibatch gradient
     $$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$
  3. Update the parameters
     $\theta \leftarrow \theta + \eta \nabla L(\theta; D')$
  4. Repeat until the validation loss stops improving.⋆

* In practice, sample without replacement until the training set is exhausted (one epoch).

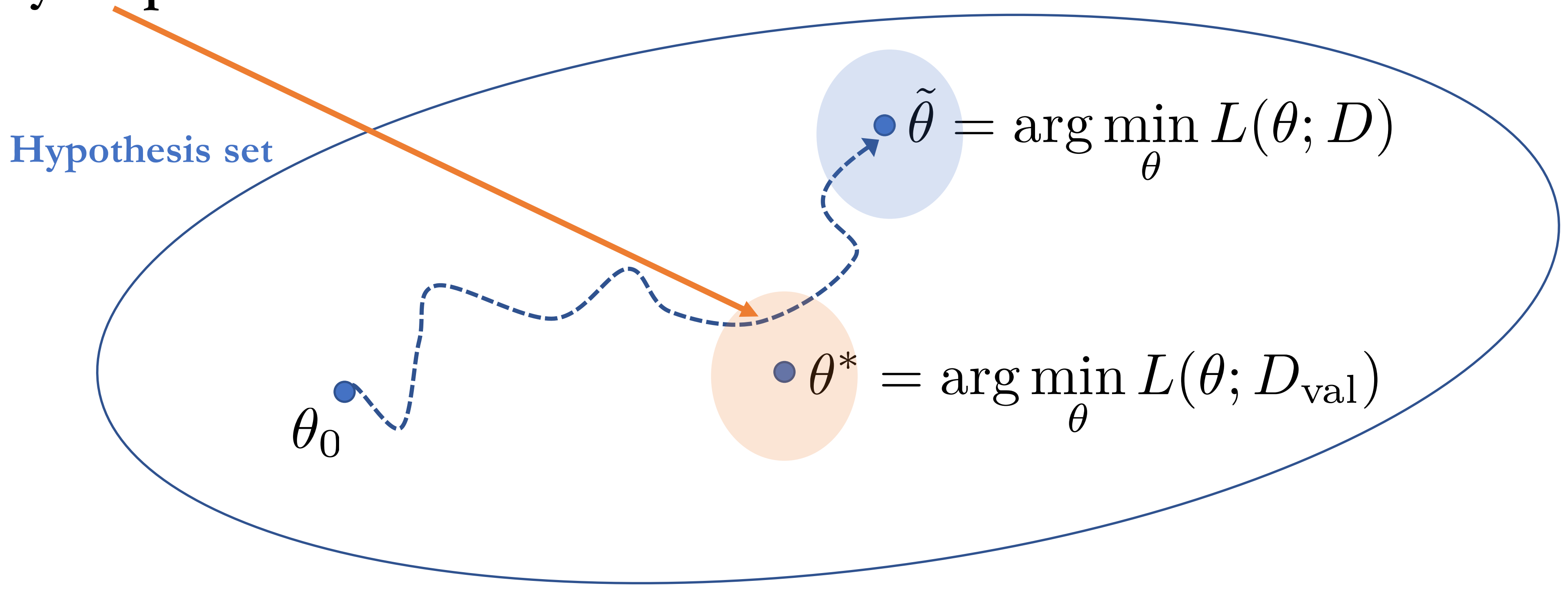
⋆ This is called early-stopping which prevents the neural network from overfitting to training examples.

Stochastic Gradient Descent – Early Stopping

- Stochastic gradient descent in practice
    1. Grab a random subset of M training examples
    2. Compute the minibatch gradient
    3. Update the parameters
    4. **Repeat until the validation loss stops improving.**
- An efficient way to prevent overfitting
    - Overfitting: the training loss is low, but the validation loss is not.
    - The most serious problem in statistical machine learning.
    - Early-stop based on the validation loss

Stochastic Gradient Descent – Early Stopping

- An efficient way to prevent overfitting
  - Overfitting: the training loss is low, but the validation loss is not.
  - The most serious problem in statistical machine learning.
  - **Early-stop** based on the validation loss



Hypothesis set
$\tilde{\theta} = \arg\min_{\theta} L(\theta; D)$
$\theta^* = \arg\min_{\theta} L(\theta; D_{\text{val}})$
θ_0

Stochastic Gradient Descent – Adaptive Learning Rate

- Stochastic gradient descent in practice
  1. Grab a random subset of M training examples $D' = \{(x_1, y_1), \ldots, (x_{N'}, y_{N'})\}$
  2. Compute the minibatch gradient
  3. Update the per-parameter learning rate η_θ
  4. Update the parameters
     $$\theta \leftarrow \theta - \eta_\theta \frac{\partial L'}{\partial \theta}$$
  5. Repeat until the validation loss stops improving.
- Adaptive learning rate: Adam [Kingma&Ba, 2015], Adadelta [Zeiler, 2015], and many more…
  - Approximately re-scale parameters to improve the conditioning of the Hessian.

Supervised Learning with Neural Networks

1. How do we decide/design a **hypothesis set**?
   - Design a network architecture as a directed acyclic graph
2. How do we decide a **loss function**?
   - Frame the problem as a conditional distribution modelling
   - The per-example loss function is a negative log-probability of a correct answer
3. How do we **optimize** the loss function?
   - Automatic backpropagation: no manual gradient derivation
   - Stochastic gradient descent with early stopping [and adaptive learning rate]

In the next lecture,

- We will study the entire cycle for text classification with a neural network.