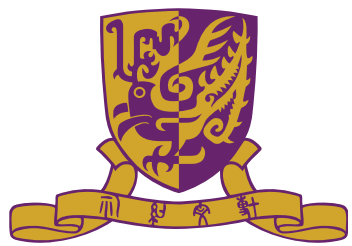
DDA5001 Machine Learning

Adam &
Introductory Deep Learning

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen



Recap: Stochastic Gradient Descent (SGD)

Consider $\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^n \ell_i(\boldsymbol{\theta})$, where n is very large.

SGD:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k).$$

SGD with Momentum:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \boldsymbol{m}_k \\ \boldsymbol{m}_k &= \mu_k \nabla \ell_{i_k}(\boldsymbol{\theta}_k) + \beta_k \boldsymbol{m}_{k-1}\end{aligned}$$

Choice of learning rate μ_k:

- Decaying: $\frac{c}{\sqrt{k}}$, step-decay, or cosine scheduler.

Adam Family

Introductory Deep Learning

Adaptive Learning Rate Method: AdaGrad

Motivations:

- All the optimization algorithms we studied now assign the same learning rate to all coordinates in $\boldsymbol{\theta}$.
- We might need different learning rates for different coordinates in $\boldsymbol{\theta}$.

AdaGrad is an important method that utilizes adaptive learning rate.

- The idea: Normalize each gradient coordinate by the past gradients.
- It has the following update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{g}_k}{\sqrt{\sum_{t=1}^{k} \boldsymbol{g}_t^2}},$$

  where $\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is the stochastic gradient. Vector division, square root, and square operations are all element-wise.
- AdaGrad penalizes the gradient coordinate when it has too large value, while increase the small one for exploration.

AdaGrad reduces the burden for tuning learning rate, and it is often more reliable than SGD.

Adam: Momentum Meets Adaptive Learning Rate

Observations:

- SGD with momentum can improve convergence speed.
- Adaptive learning rate like AdaGrad can reduce learning rate tuning and stablize convergence.

How about combine these two techniques? $\rightsquigarrow$Adam method.

- Adam uses a moving averaging for updating the momentum and adaptive learning rate.

Momentum update:

$$\boldsymbol{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k, \quad \text{with} \quad \boldsymbol{m}_0 = 0.$$

Adaptive learning rate update:

$$\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2, \quad \text{with} \quad \boldsymbol{v}_0 = 0.$$

$\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$ is the stochastic gradient.

Rule of thumb: $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Understanding Moving Averaging

If we expand the update of momentum and adaptive learning rate in Adam, we will have

$$\boldsymbol{m}_k = \beta_1^k \boldsymbol{m}_0 + \beta_1^{k-1}(1-\beta_1)\boldsymbol{g}_1 + \cdots + \beta_1(1-\beta_1)\boldsymbol{g}_{k-1} + (1-\beta_1)\boldsymbol{g}_k,$$

and

$$\boldsymbol{v}_k = \beta_2^k \boldsymbol{v}_0 + \beta_2^{k-1}(1-\beta_2)\boldsymbol{g}_1^2 + \cdots + \beta_1(1-\beta_2)\boldsymbol{g}_{k-1}^2 + (1-\beta_2)\boldsymbol{g}_k^2,$$

Interpretations: The moving averaging exponentially decays the importance of the historical records, while emphasizes more on the most recent ones.

Adam: Algorithmic Procedure

Adam

1. Compute a stochastic gradient $\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k)$;
2. Update momentum: $\boldsymbol{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\boldsymbol{g}_k$;
3. Update adaptive learning rate $\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\boldsymbol{g}_k^2$;
4. Bias-corrected momentum: $\boldsymbol{m}_k = \boldsymbol{m}_k/(1-\beta_1^k)$;
5. Bias-corrected adaptive learning rate: $\boldsymbol{v}_k = \boldsymbol{v}_k/(1-\beta_2^k)$;
6. Update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{m}_k}{\sqrt{\boldsymbol{v}_k + 10^{-8}}}.$$

Repeat until convergence.

- The bias-correction term is used to make $\boldsymbol{m}_k$ and $\boldsymbol{v}_k$ unbiased estimators for $\mathbb{E}[\boldsymbol{g}_k]$ and $\mathbb{E}[\boldsymbol{g}_k^2]$.
- Adam is widely used in practice for training NN.

Weight Decay Issue in Adam

Weight decay regularization is widely utilized in NN training, since otherwise it easily has overfitting due to strong representation power of NN.

- Therefore, Adam is often used together with weight decay.
- However, how people use it is through changing the stochastic gradient:

$$\boldsymbol{g}_k = \nabla \ell_{i_k}(\boldsymbol{\theta}_k) + \lambda \boldsymbol{\theta}_k.$$

  Recall that the gradient of weight decay term $\frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$ is $\lambda\boldsymbol{\theta}$.

Issues:

- $\boldsymbol{g}_k$ is not directly used for updating, it is used to update momentum and also normalized by adaptive learning rate.
- This makes it unclear whether we have weight decay nor not.
- It is not equivalent to the ℓ_2 regularization on $\boldsymbol{\theta}$ directly we studied in the regularization part.

AdamW: Decoupled Weight Decay

Idea: Decouple the weight decay term and use the true ℓ_2-regularization.
$\rightsquigarrow$ the AdamW method.

- All the steps of AdamW keeps the same as that of the Adam, except for the update:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \mu_k \frac{\boldsymbol{m}_k}{\sqrt{\boldsymbol{v}_k + 10^{-8}}} - \mu_k \lambda \boldsymbol{\theta}_{k-1}.$$

- This additional $\lambda \boldsymbol{\theta}_{k-1}$ corresponds to the gradient of the weight decay term added to training loss function directly, namely, for NN training

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2,$$

  AdamW applies Adam for $\min \mathcal{L}$ while GD for $\min$ the weight decay regularization.

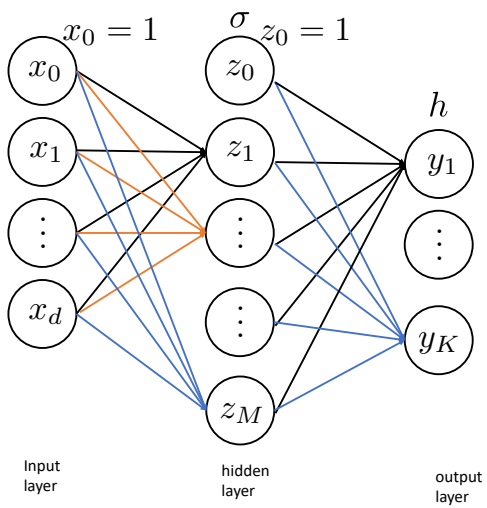
- AdamW is truly applying ℓ_2-regularization directly on $\boldsymbol{\theta}$. Thus, for huge NN training problems, AdamW often has better generalization than Adam, as better regularization improves generalization.

Experiments: Setup

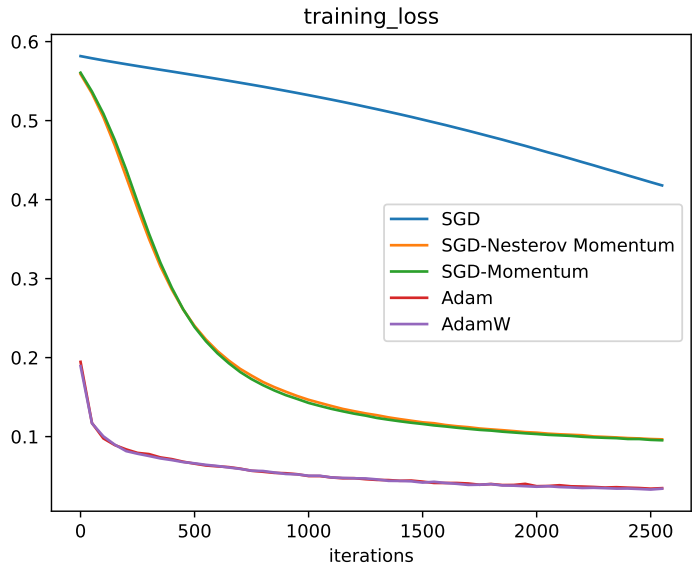
We now conduct experiments on SGD, SGD momentum, SGD nesterov momentum, Adam, AdamW for training NN.

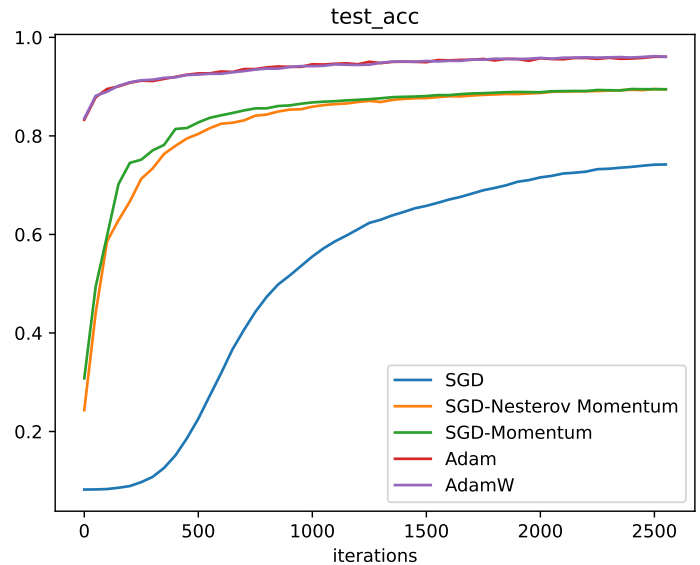
- Task: Classification on the MNIST dataset:

- Model: One hidden layer (two layer) NN, with $M = 64$, number of classes $K = 10$, σ is ReLU, h is softmax.



Experiments: Results[1]





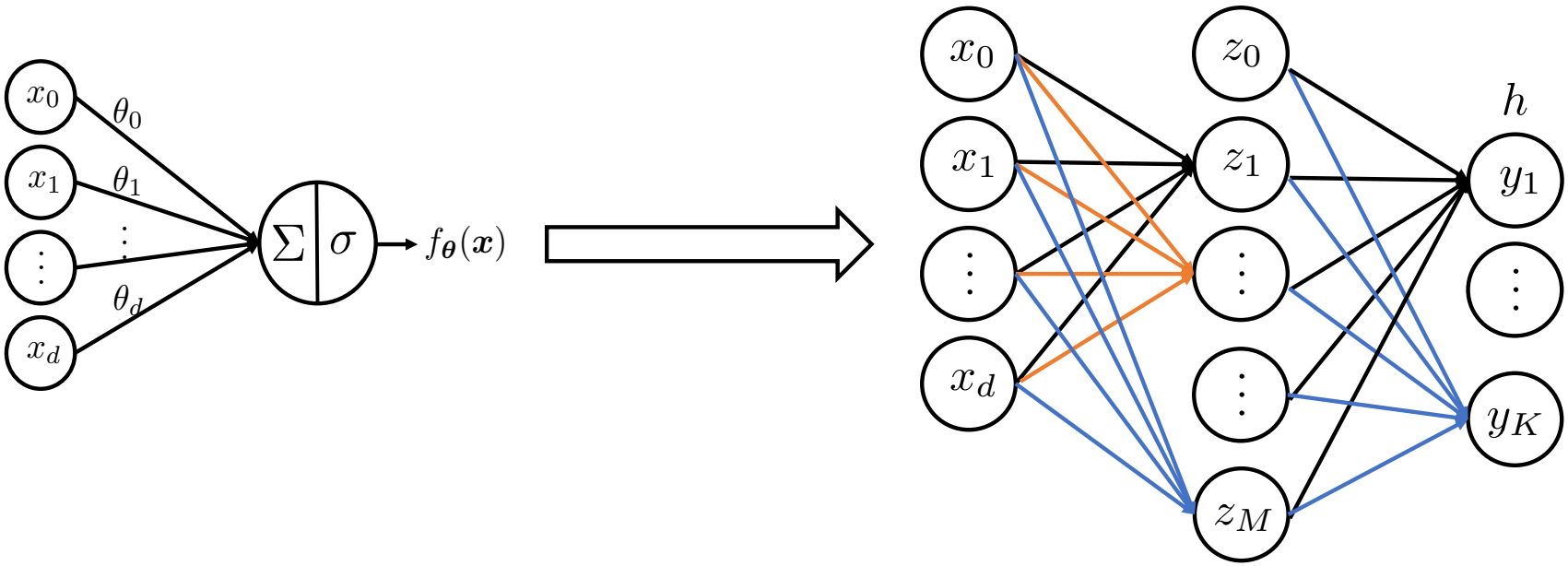
- SGD is slow.
- Adam family has the best performance, illustrating why they are ubiquitous in practice.

[1] Code is available at `https://www.kaggle.com/code/jonery/lecture-demo`. You need to register Kaggle to use it.

Adam Family

Introductory Deep Learning

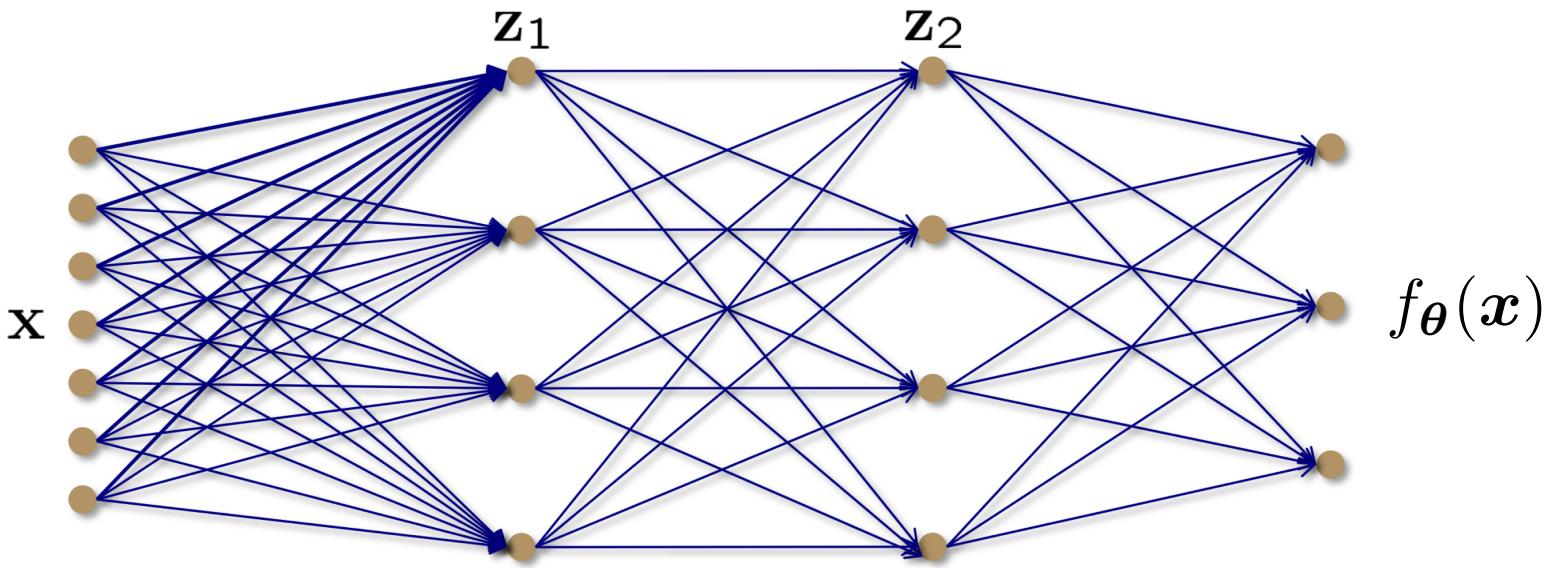
We Extended Single Layer to Two Layer



One quick idea: Why not go deeper (more hidden layers)?

Deep Learning: Multi-Layer Neural Networks

- The framework of one hidden layer neural network, along with the backpropagation, can easily be extended to networks with multiple hidden layers.



- This model has had a resurgence in recent years under the name deep learning.

Deep Learning: Deep Feed-Forward Networks

- Deep feed-forward networks are the essential deep learning models. It is a direct generalization of the former two-layer neural network model.
- Given input $\boldsymbol{x} \in \mathbb{R}^d$, the deep feed-forward neural network model is

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{W}^L \sigma(\boldsymbol{W}^{L-1} \cdots \sigma(\boldsymbol{W}^2 \sigma(\boldsymbol{W}^1 \boldsymbol{x}))))$$

- $\boldsymbol{W}^l \in \mathbb{R}^{d_l \times d_{l-1}}$ for $l = 1, \ldots, L$, where $d_0 = d$ and $d_L = K$
- $l = 1, \ldots, L$ are call layers, while $\max_l d_l$ are called width.
- Parameters to learn: $\boldsymbol{\theta} = (\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L)$.
- σ is called activation function, remaining the same for all hidden layers.
- h function in the final layer depends on applications.
- $\boldsymbol{z}^l = \sigma(\boldsymbol{W}^l \boldsymbol{z}^{l-1})$ for $l = 1, \ldots, L$, where $\boldsymbol{z}^0 = \boldsymbol{x}$.

The Learning Problem

- Regression

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{y}_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\|_2^2 \right\}$$

- Classification

$$\min_{\boldsymbol{\theta}} \left\{ \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i^\top \log(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \right\}$$

- It has exactly the same interpretation as two-layer NN, applying LS or LR in the last year using learned feature $\boldsymbol{z}$ rather than sample $\boldsymbol{x}$.
- The only difference is now $\boldsymbol{z} = \sigma(\boldsymbol{W}^{L-1} \cdots \sigma(\boldsymbol{W}^2 \sigma(\boldsymbol{W}^1 \boldsymbol{x})))$, which is the feature extracted through many hidden layers rather than just one hidden layer.
- Learning the feature $\boldsymbol{z}$ from $\boldsymbol{x}$ is also called representation learning. Hence ,deep learning is representation learning plus final layer linear regression or classification.

Backpropagation

Taking ℓ_2-loss function as example

$$\ell_i(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - h(\boldsymbol{W}^L \sigma(\boldsymbol{W}^{L-1} \cdots \sigma(\boldsymbol{W}^2 \sigma(\boldsymbol{W}^1 \boldsymbol{x}_i))))\|_2^2$$

We omit subscription i in the sequel.

Forward pass: Feed $\boldsymbol{x}$ using the current up-to-date $\boldsymbol{\theta}$ to compute and store the activation values

$$\boldsymbol{z}^0 = \boldsymbol{x},$$

$$\boldsymbol{z}^1 = \sigma(\boldsymbol{W}^1 \boldsymbol{z}^0)$$

$$\vdots$$

$$\boldsymbol{z}^{L-1} = \sigma(\boldsymbol{W}^{L-1} \boldsymbol{z}^{L-2})$$

$$\boldsymbol{z}^L = h(\boldsymbol{W}^L \boldsymbol{z}^{L-1})$$

Backpropagation

Rewrite the objective function

$$\ell(\boldsymbol{\theta}) = \|\boldsymbol{y}_i - h(\boldsymbol{W}^L \sigma(\boldsymbol{W}^{L-1} \cdots \underbrace{\sigma(\boldsymbol{W}^l \boldsymbol{z}^{l-1})}_{\boldsymbol{z}^l}))\|_2^2$$

where

$$\frac{\partial \ell}{\partial \boldsymbol{W}^l(m,j)} = \frac{\partial \ell}{\partial \boldsymbol{z}_m^l} \cdot \sigma'\left((\boldsymbol{w}_m^l)^\top \boldsymbol{z}^{l-1}\right) \cdot \boldsymbol{z}^{l-1}[j]$$

By chain rule, we have

$$\frac{\partial \ell}{\partial \boldsymbol{z}^l} = \frac{\partial \ell}{\partial \boldsymbol{z}^{l+1}} \times \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{z}^l}$$

Backward pass: One first computes $\frac{\partial \ell}{\partial \boldsymbol{z}^L}$, then $\frac{\partial \ell}{\partial \boldsymbol{z}^l}$, ..., and finally $\frac{\partial \ell}{\partial \boldsymbol{z}^1}$. A generalized procedure for two-layer's case.

- After getting the stochastic gradient, we can apply AdamW for the training.

How Many Layers and Nodes? Issue of Overfitting

- Choosing the number of hidden layers and number of nodes in each layer is more art than science.
- In general, emperical performance suggests that it is better to have too many parameters (i.e., deep and wide) rather than too few.
- Too many parameters and too strong representation power easily leads to overfitting.
- One main aspect of deep learning is to avoid overfitting.

Ways to avoid overfitting:

- Regularization.
- Early stopping (validation).
- ...

Philosophical summary: Deep learning is to give a strong enough model that can easily overfit any data. The task of learning is to learn the model parameters by simultaneously fitting the data and searching the right complexity ($d^{\star}_{\text{VC}}$) of the model (by ways of avoiding overfitting).

Regularization

- To avoid overfitting, regularization is crucial. The usually utilized regularization in deep learning is wight decay.
- The learning problem becomes

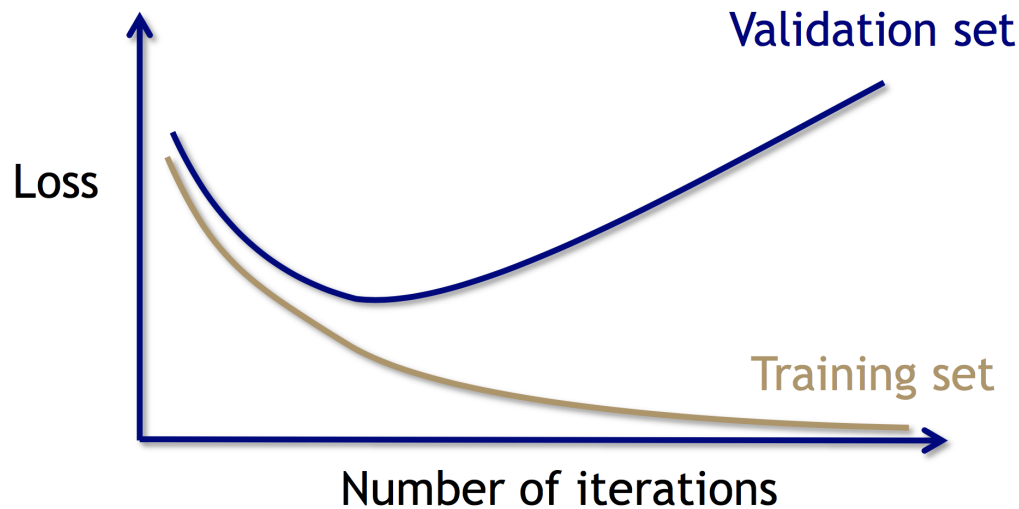
$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2.$$

- This encourages small (or even zero) weights; see regularization lecture for interpretation.
- If we constrain some weights to be small or even zero, the entire network has a much less complex model, thus reducing overfitting.
- Note that Adam does not utilizes weight decay properly. We often use AdamW.

Early Stopping (Validation)

For a generic optimization problem, running AdamW for more iterations would always be better.

But in the training of deep neural networks, we have another way of stopping the learning algorithm.



- Using validation dataset to stop the algorithm early.
- Early stopping stops the optimizer when the validation error is going to increase, thus avoiding overfitting.

The Success of Deep Learning

Deep learning is very successful in computer vision (early days) and natural language processing (nowadays large language models especially), etc. The supporting elements are:

- Automatic differentiation for implementing BP for almost any network.
- The optimization algorithms, e.g., Adam family.
- The computational power (GPUs).
- Techniques to avoid overfitting.
- Very large and carefully designed dataset.

$\rightsquigarrow$We are going to study several other architectures other than feed-forward NN.

Other Architectures: Convolutional Neural Networks

Convolutional Neural Networks

One well-known NN architecture is the convolutional neural network (CNN).

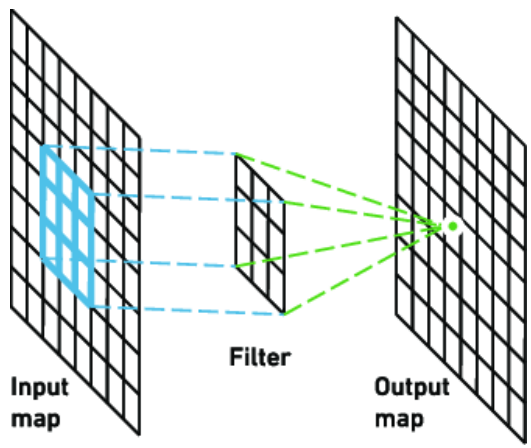
- It makes the explicit assumption that the input is an image.
- It constrains the structure of the network to make learning the weights scalable to high-resolution images and less sensitive to rotation and shifts of pixels, i.e., through convolution.
- Convolution is a fundamental concept in signal and image processing.
- The most widely used activation function in CNN is ReLU:

$$\sigma(t) = \max(0, t).$$

Convolutional layer

- In signal and image processing, the convolution operator is often analytical.
- In deep learning, it is learned. The weight matrix in CNN is the convolution operator, i.e., the parameters of network to be learned. It is also called filter.
- The l-th convolutional layer can be regarded as $\boldsymbol{W}^l \boldsymbol{z}^{l-1}$ in the deep feed-forward neural network. But it does not use matrix-vector multiplication, it uses convolution, which is quite efficient even when the image dimension is very large.

The output of these filters $\boldsymbol{W}^l \boldsymbol{z}^{l-1}$ are then passed through a ReLU activation $\sigma(\boldsymbol{W}^l \boldsymbol{z}^{l-1})$.

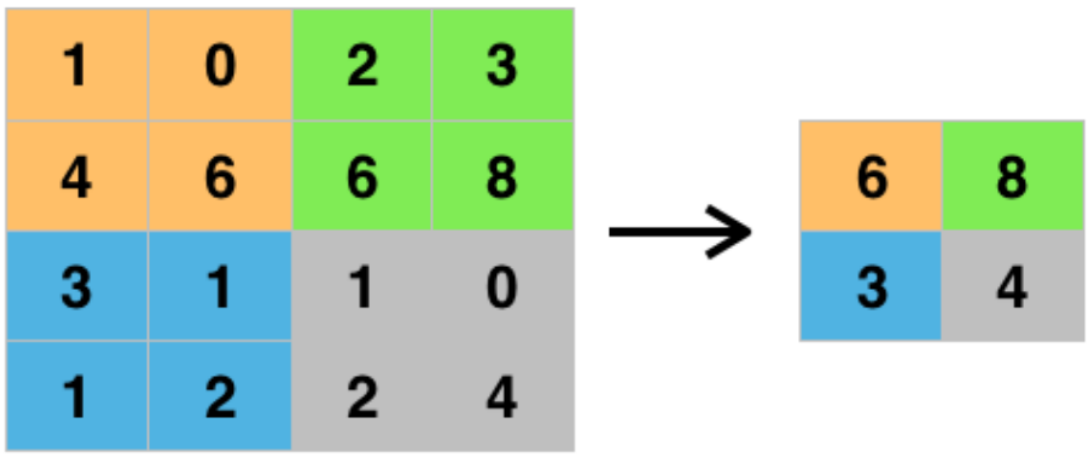


Pooling Layer

The other key concept in CNNs is pooling (subsampling).

- Given the output of a filter, we can downsample the output to produce a smaller number of pixels for the next layer.

Most common choice is known as max pooling:

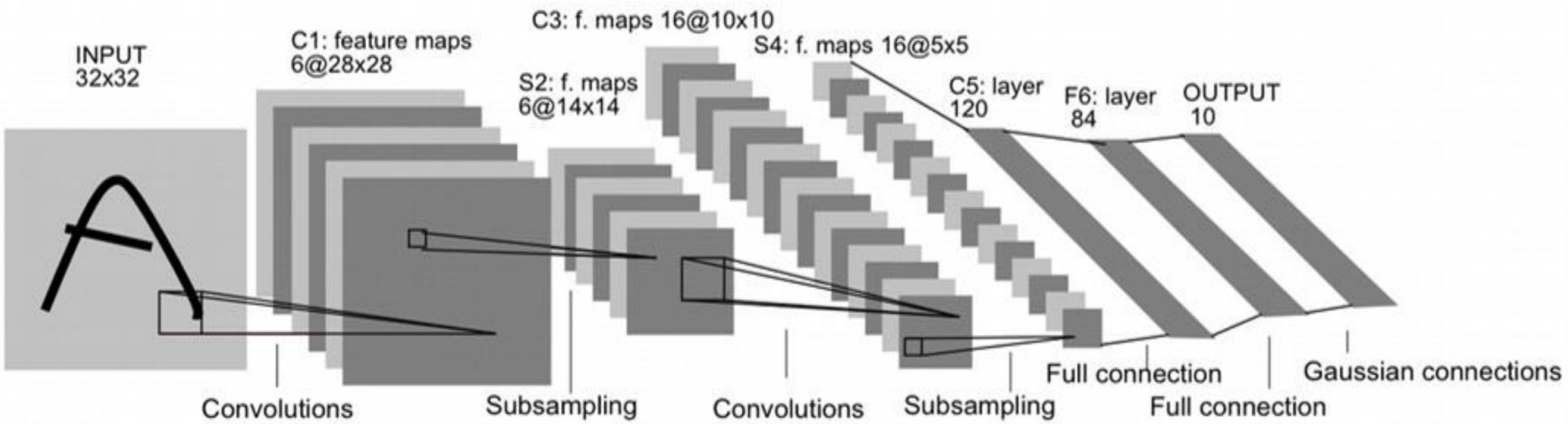


Interpretation: The precise location of a feature is not important.

A CNN will typically have a pooling layer after each convolution layer.

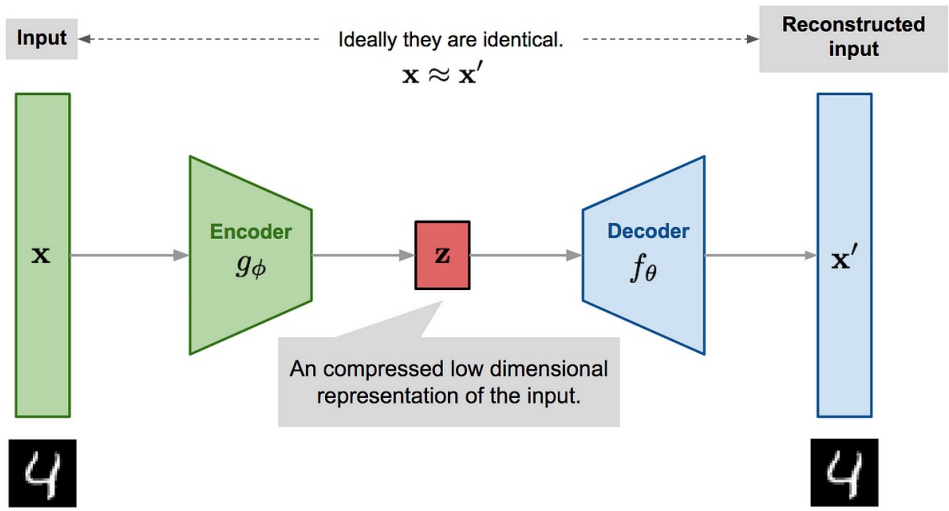
Full CNN

Yann LeCun (1998):



More Architectures

- Autoencoders.



  - Input and output are the same, but the number of hidden nodes is constrained to be small relative to input dimension.
  - Way of doing dimensionality reduction $\rightsquigarrow$ generalization of PCA.

- And many others...

Keep one thing in mind, deep learning is mainly to learn representations (i.e., z) using NN, and doing linear model in the final layer.

$\rightsquigarrow$ Next lecture: Transformer.