



CMP9137M Advanced Machine Learning


Lecture 4: Classification and Optimization in CNNs

<https://attendance.lincoln.ac.uk>

Access Code:751187

School of Computer Science

Laboratory of Vision Engineering

 Dr. Lei Zhang

 lzhang@Lincoln.ac.uk



Objectives

- To understand softmax and the cross-entropy loss function for the softmax function in the classification
- To know the optimization process with different optimizers in the training
- To know the evolution of the CNNs

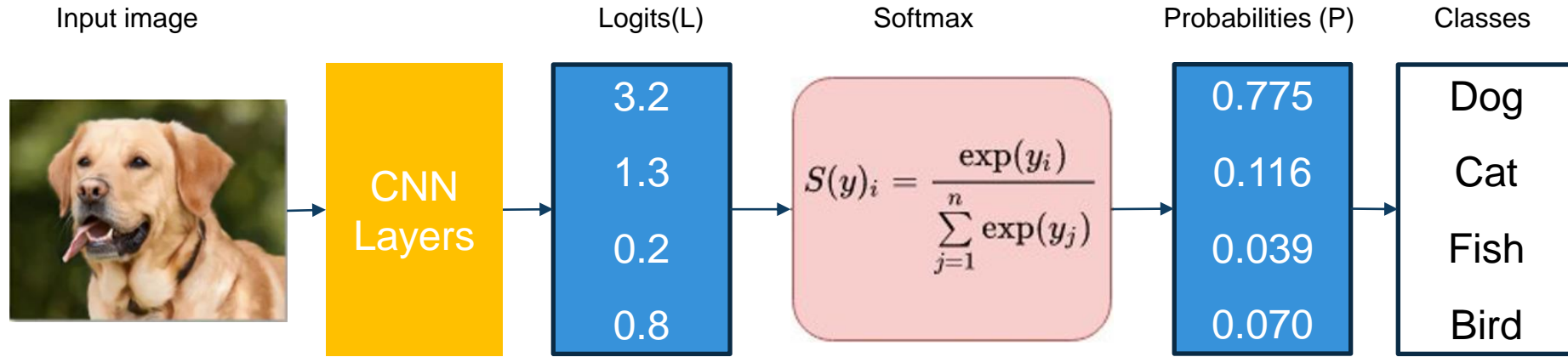


Outline

Covering:

- **Softmax + cross-entropy loss for classification**
(Softmax activation, Softmax with cross-entropy loss, image classification in practice)
- **Optimization methods**
(Gradient descent, Learning rater, Full Gradient descent, Stochastic Gradient Descent (SGD), Momentum and adaptive method)
- **Classic CNN architectures**
(LeNet, AlexNet, Zfnet, VGG, ResNet, and their evolution)

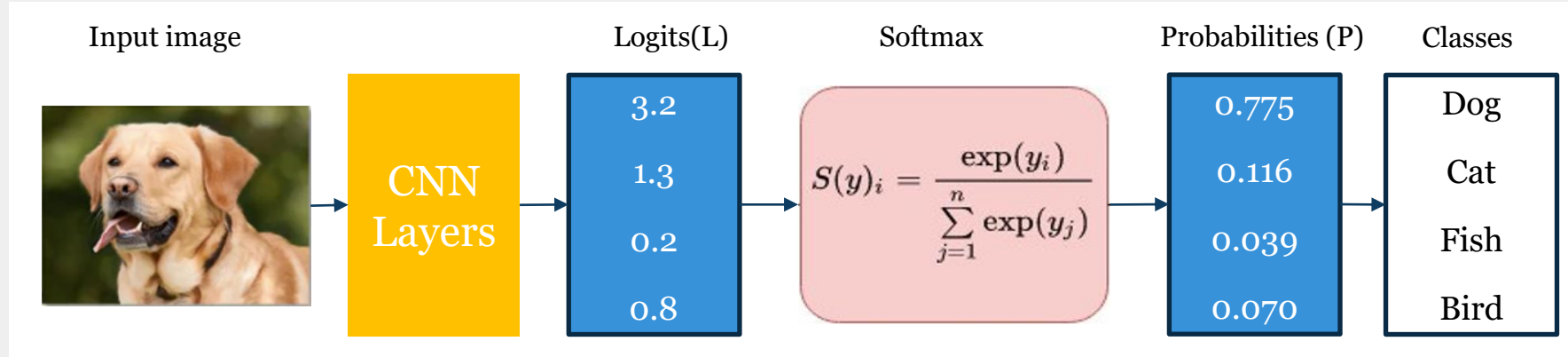
CNN-Softmax function for classification



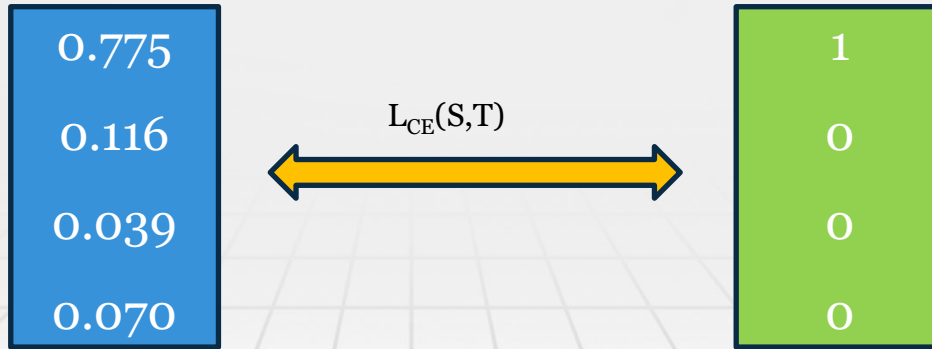
It applies the standard exponential function to each element y_i and normalize these values by dividing by the sum of all these exponentials;

$$p \in [0, 1] \text{ and } \sum_{j=1}^n p_n = 1$$

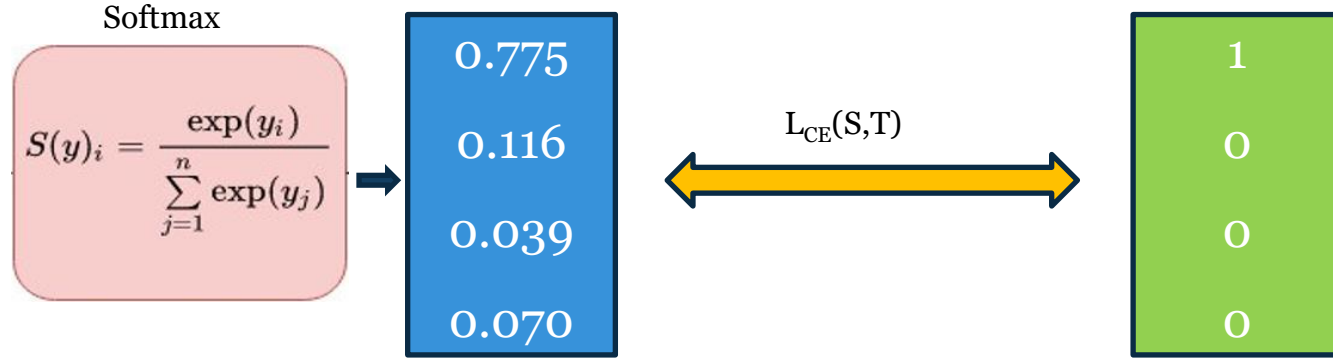
CNN-Cross Entropy Loss function



Cross-Entropy is used to measure the distance between the probabilities (P) and truth values.



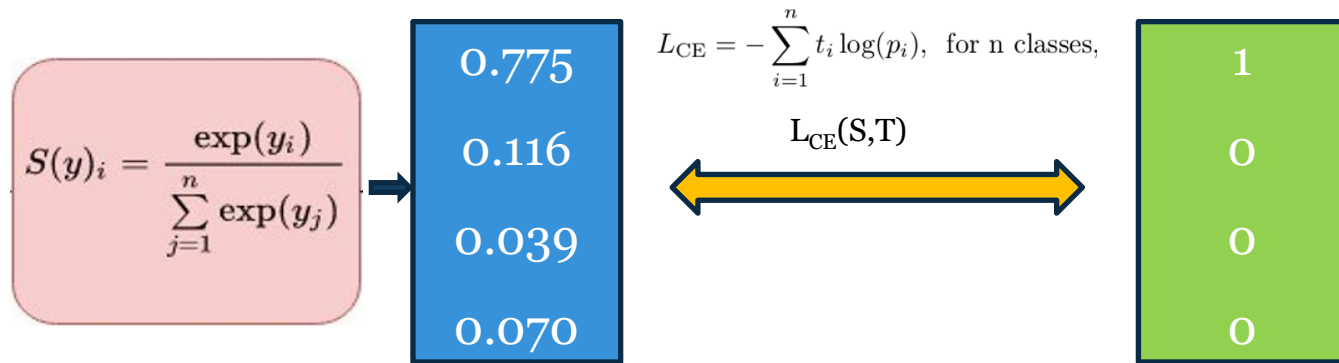
CNN.Cross Entropy Loss



$$L_{CE} = - \sum_{i=1}^n t_i \log(p_i), \text{ for } n \text{ classes,}$$

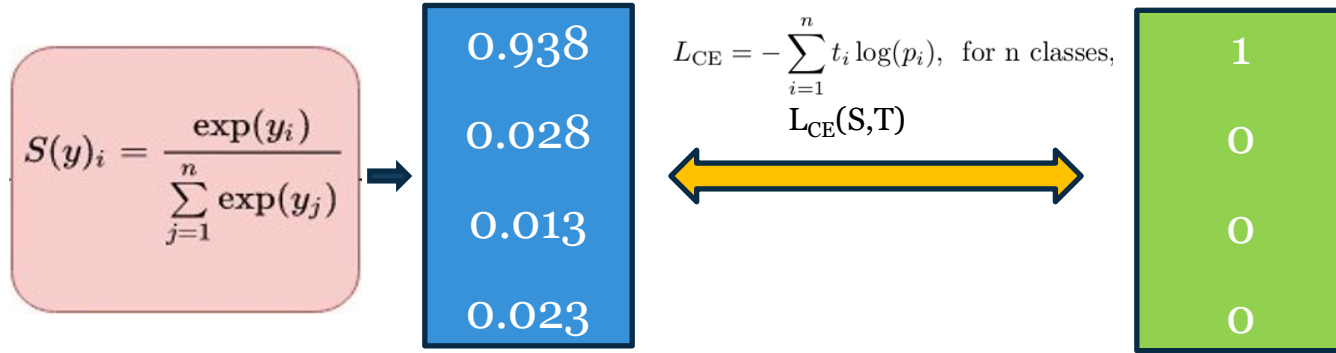
where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

CNN.Cross Entropy Loss



$$\begin{aligned} L_{CE} &= - \sum_{i=1} T_i \log(S_i) \\ &= - [1 \log_2(0.775) + 0 \log_2(0.116) + 0 \log_2(0.039) + 0 \log_2(0.070)] \\ &= - \log_2(0.775) \\ &= 0.3677 \end{aligned}$$

CNN.Cross Entropy Loss



$$\begin{aligned} L_{CE} &= -1 \log_2(0.936) + 0 + 0 + 0 \\ &= 0.095 \end{aligned}$$

CNN-Softmax function for classification

$$S(z)_c = \frac{e^{z_c}}{\sum_{d=1}^D e^{z_d}} \text{ for } c = 1 \dots C$$

$$S(z)_c \in [0,1] \text{ and}$$

$$\sum_{c=1}^C S(z)_c = 1$$

- The softmax function $S(z)_c$ takes as input a C -dimensional vector z and outputs a C -dimensional vector of values between 0 and 1.
- This function is a normalized exponential.
- The denominator $\sum_{d=1}^D e^{z_d}$ acts as a regularizer to make sure that $\sum_{c=1}^C S(z)_c = 1$

CNN-Derivative of the softmax function

$$S(z)_c = \frac{e^{z_c}}{\sum_{d=1}^D e^{z_d}} \text{ for } c = 1 \dots C$$

$$S(z)_c \in [0,1] \text{ and}$$

$$\sum_{c=1}^C S(z)_c = 1$$

$$y_c = S(z)_c = \frac{e^{z_c}}{\sum C} \text{ for } c = 1 \dots C$$

$$\text{where } \sum C = \sum_{d=1}^D e^{z_d}$$

- **Quotient Rule:** $h(x) = \frac{f(x)}{g(x)}$ then $h'(x) = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$

This derivative $\partial y_i / \partial z_j$ of the output y of the softmax function with respect to its input z can be calculated as:

If $i=j$:

$$\begin{aligned} \frac{\partial y_i}{\partial z_i} &= \frac{\partial \frac{e^{z_i}}{\sum_{d=1}^D e^{z_d}}}{\partial z_i} = \frac{e^{z_i} \cdot (\sum_{d=1}^D e^{z_d}) - e^{z_i} \cdot e^{z_i}}{(\sum_{d=1}^D e^{z_d})^2} \\ &= \frac{e^{z_i} \cdot (\sum C) - e^{z_i} \cdot e^{z_i}}{(\sum C)^2} = \frac{e^{z_i} \cdot (\sum C) - e^{z_i} \cdot e^{z_i}}{(\sum C) \cdot (\sum C)} \\ &= \frac{e^{z_i} \cdot ((\sum C) - e^{z_i})}{(\sum C) \cdot (\sum C)} = \frac{e^{z_i}}{(\sum C)} \cdot \frac{(\sum C) - e^{z_i}}{(\sum C)} \\ &= y_i \cdot (1 - y_i) \end{aligned}$$

Quotient Rule

If $i \neq j$:

$$\begin{aligned} \frac{\partial y_i}{\partial z_j} &= \frac{\partial \frac{e^{z_i}}{\sum_{d=1}^D e^{z_d}}}{\partial z_j} = \frac{0 - e^{z_i} \cdot e^{z_j}}{(\sum_{d=1}^D e^{z_d})^2} = \frac{e^{z_i}}{(\sum C)} \cdot \frac{e^{z_j}}{(\sum C)} \\ &= y_i \cdot y_j \end{aligned}$$

Note that if $i=j$ this derivative is similar to the derivative of the sigmoid function.

CNN-Cross Entropy Loss. softmax + cross entropy loss

- The combination of cross entropy loss and softmax as last-layer's activation is the standard configuration in classification.
- The cross-entropy error function over a batch of multiple samples of size n can be calculated as:

$$L_{CE}(Y,T)=-\sum_{i=1}^n \sum_{c=1}^C t_{ic} \cdot \log(y_{ic})$$

where t_{ic} is 1 if and only if sample i belongs to class c , and y_{ic} is the output probability that sample i belongs to class c .

The softmax loss layer computes the multinomial logistic loss of the softmax of its inputs. It's conceptually identical to a softmax layer followed by a multinomial logistic loss layer, but provides a more numerically stable gradient.

- To improve the numerical stability, many ML libraries combine the two as one layer.

CNN-Derivative of the cross-entropy loss function for the softmax function

$$L_{CE}(Y,T) = -\sum_{i=1}^n \sum_{c=1}^C t_{ic} \cdot \log(y_{ic})$$

The derivative $\partial L / \partial z_i$ of the loss function with respect to the softmax input z_i can be calculated as:

If $i=j$: $\frac{\partial y_i}{\partial z_j} = \frac{\partial y_i}{\partial z_i} = y_i \cdot (1 - y_i)$

If $i \neq j$: $\frac{\partial y_i}{\partial z_j} = y_i \cdot y_j$

$$\begin{aligned} \frac{\partial L}{\partial z_i} &= -\sum_{j=1}^C \frac{\partial t_j \log(y_j)}{\partial z_i} = -\sum_{j=1}^C t_j \frac{\partial \log(y_j)}{\partial z_i} = -\sum_{j=1}^C t_j \frac{1}{y_j} \frac{\partial y_j}{\partial z_i} \\ &= -\frac{t_i}{y_i} \frac{\partial y_i}{\partial z_i} - \sum_{j \neq i}^C \frac{t_j}{y_j} \frac{\partial y_j}{\partial z_i} = -\frac{t_i}{y_i} y_i (1 - y_i) - \sum_{j \neq i}^C \frac{t_j}{y_j} (-y_j y_i) \\ &= -t_i + t_i y_i + \sum_{j \neq i}^C t_j y_i = -t_i + \sum_{j=1}^C t_j y_i = -t_i + y_i \sum_{j=1}^C t_j \\ &= y_i - t_i \end{aligned}$$

Logarithm Derivatives

$$f(x) = \ln(x) \text{ then } f'(x) = \frac{1}{x}$$

CNN-image Classification in practice (Code review)

Training a neural network on MNIST with Keras

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dense(10)  
])
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290

Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0

Training a CNN on MNIST with Keras

```
from tensorflow import keras
from tensorflow.keras import layers

1. model = keras.Sequential(
2.     [
3.         keras.Input(shape=input_shape),
4.         layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
5.         layers.MaxPooling2D(pool_size=(2, 2)),
6.         layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
7.         layers.MaxPooling2D(pool_size=(2, 2)),
8.         layers.Flatten(),
9.         layers.Dropout(0.5),
10.        layers.Dense(num_classes, activation="softmax"),
11.    ]
12.)

batch_size = 128
epochs = 15
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
)
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010
Total params: 34,826		
Trainable params: 34,826		
Non-trainable params: 0		

... https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/vision/ipynb/mnist_convnet.ipynb

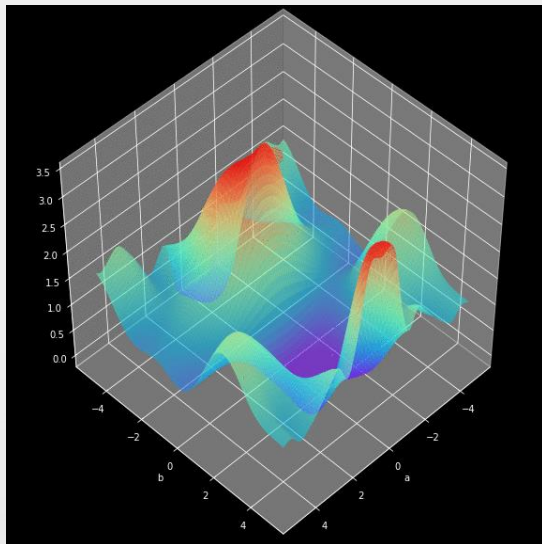


Outline

Covering:

- **Softmax + cross entropy loss for classification**
(softmax activation, Softmax with cross entropy loss, image classification in practice)
- **Optimization methods**
(Gradient descent, Learning rate, Full Gradient descent, Stochastic Gradient Descent (SGD), Momentum and adaptive method)
- **Classic CNN architectures**
(LeNet, AlexNet, Zfnet, VGG, ResNet, and their evolution)

How do the neural networks learn-The training procedure (last week)



- ① Take a set of training examples you wish the network to learn from
- ② Set up the network with N input units fully connected to M non-linear hidden units via connections with weights w_{ij} , which in turn are fully connected to P outputs via connections with weights w_{jk}
- ③ Generate random initial weights, e.g. from range $[-0.01, +0.01]$
- ④ Select an appropriate error function $E(w_{jk})$ and learning rate η
- ⑤ For each training example:
 - ▶ **Forward** pass: Calculate predicted output(s) a_k
 - ▶ **Backward** pass: Calculate weight updates Δw_{ik} and Δw_{ji}
 - ▶ Update network weights ($w_{new} = w_{old} + \Delta w$)
- ⑥ Repeat step 5 until the network error is 'small enough'

CNN-Optimization with Gradient descent

Problem:

$$\min_w L(w)$$

Gradient descent one of the most basic optimisation methods

Iterative Solution:

$$w_{k+1} = w_k - \eta_k \nabla L(w_k)$$

w_{k+1} is the updated weight after the k_{th} iteration

w_k is the initial weight before the k_{th} iteration

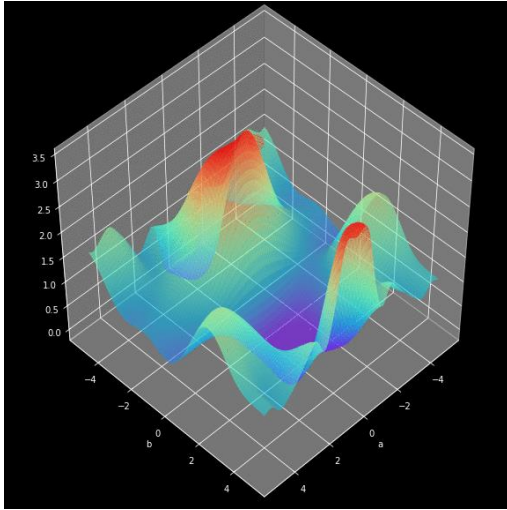
η_k is the step size (Learning rate)

$\nabla L(w_k)$ is the gradient of L

Gradient descent optimizes the parameters used in the model by computing the gradient of the loss with respect to parameters. This gradient is then used to continually improve the parameters step by step.

CNN-Optimization with Gradient descent-principle

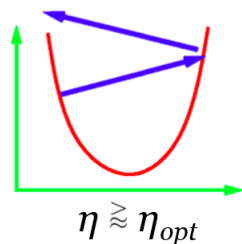
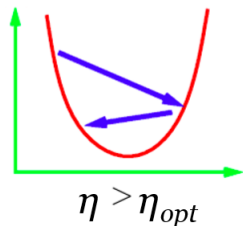
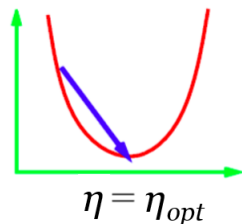
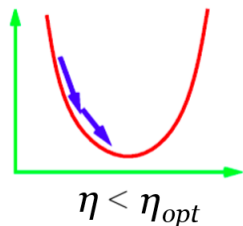
$$w_{k+1} = w_k - \eta_k \nabla L(w_k)$$



- Aims to find the lowest point (valley) of the optimization function.
- The actual direction to this valley is not known. Look locally, and therefore the direction of the negative gradient is the best information that we have.
- Taking a small step in that direction, Once we have taken the small step, we again compute the new gradient and again move a small amount in that direction, till we reach the valley

CNN-Optimization with Gradient descent. **Learning rate**

$$w_{k+1} = w_k - \eta_k \nabla L(w_k)$$



divergence

- Generally, we don't know the value of the optimal learning rate. It is generally very difficult (or impossible) to get a learning rate that would directly take us to the minimum.
- A few different scenarios that can occur. (learning rate is too low, a little larger than the optimal).
- Standard practice is to try a bunch of values on a log-scale and then use the best one.

Step sizes for 1D Quadratic

CNN-Optimization with **Full-batch gradient descent**

$$\begin{aligned}\nabla L &= \nabla \frac{1}{n} \sum_{i=1}^n \text{loss}(f(x_i), y_i) \\ &= \frac{1}{n} \sum_{i=1}^n \nabla \text{loss}(f(x_i), y_i)\end{aligned}$$

$$w_{k+1} = w_k - \eta_k \nabla L(w_k)$$

- The full-batch gradient descent fully uses the available training data in each iteration. It is the total loss over all instances. However, it is especially inefficient if there is duplicated training data.
- Since computing the gradient is by far the most expensive part of gradient descent, it makes sense to try to make this more efficient.
- To save time, it uses a part of the data to estimate the gradient.

CNN-Optimization with Stochastic Gradient Descent (SGD)

A i_{th} data point x_i and label y_i are sampled uniformly from the training set. The true gradient ∇L is then estimated using only this data point and label:

$$\nabla l_i = \nabla \text{loss}(x_i, y_i, w_k)$$

l_i is a unbiased estimator of L which is mathematically written as:

$$\mathbb{E}[\nabla l_{i(w_k)}] = \nabla L(w_k)$$

Theorem: If k estimators all produce unbiased estimates $\tilde{X}_1, \dots, \tilde{X}_k$ of X , then any weighted average of them is also an unbiased estimator. The full estimate is given by

$$\tilde{X} = w_1 * \tilde{X}_1 + \dots + w_k * \tilde{X}_k$$

where the sum of weights $\sum_{i=1}^k w_i = 1$ needs to be normalized.

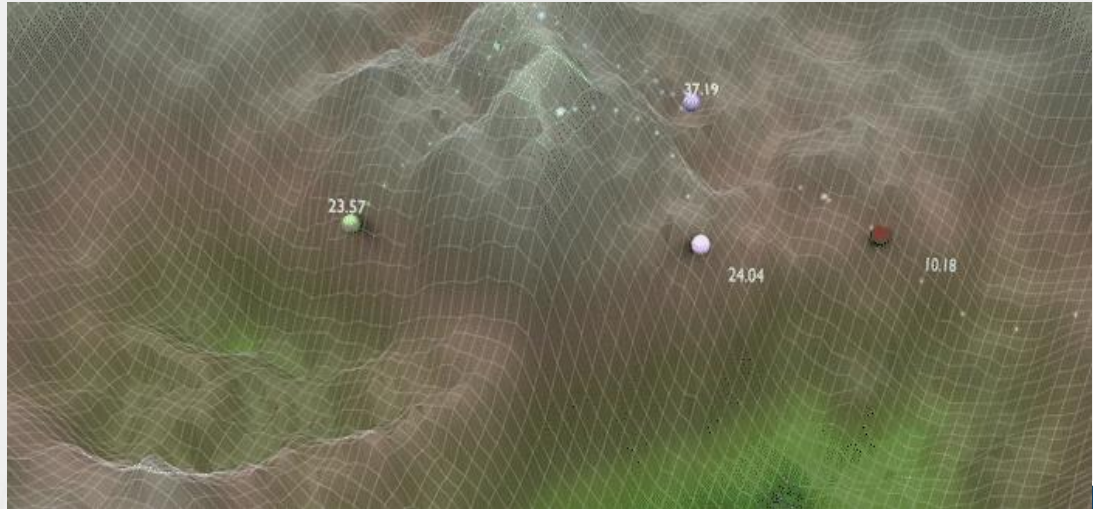
In SGD, we update the weights according to the gradient over l_i (as opposed to the gradient over the total loss L). As a result of this, the expected k -th step of SGD is the same as the k -th step of full gradient descent:

$$\mathbb{E}[w_{k+1}] = w_k - \eta_k \mathbb{E}[\nabla l_{i(w_k)}] = w_k - \eta_k \nabla L(w_k)$$

CNN-Optimization with Stochastic Gradient Descent (SGD).advantages

$$\mathbb{E}[w_{k+1}] = w_k - \eta_k \mathbb{E}[\nabla l_i(w_k)] = w_k - \eta_k \nabla L(w_k)$$

- The computations for SGD can be performed very quickly but still give us an unbiased estimate of the true gradient.
- The noise in SGD can help us avoid the shallow local minima and find a better (deeper) minima. This phenomenon is called **annealing**.
- There is a lot of redundant information across instances. SGD prevents a lot of these redundant computations.



CNN-Optimization with **Mini-batch gradient descent**

- The individual SGD estimates can have a large variance however, leading to noisy and jumpy updates.
- In mini-batching, we consider the loss over a small batch of randomly selected (B) examples instead of calculating it over just one instance. The estimated gradient is an average of all B single estimates. This reduces the noise in the step update.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \frac{1}{|B_j|} \sum_{i \in B_j} \nabla l_i(\mathbf{w}_k)$$

- These gradient calculations In mini-batching can be parallelized very well on GPUs and In Distributed network training (split a large mini-batch between the machines of a cluster and then aggregate the resulting gradients.
- Mini-batch gradient descent is typically not much slower than SGD but leads to a **more stable optimization process**.

CNN-SGD with Momentum. **Intuition**



Fig.1 SGD without momentum



Fig.2 SGD with momentum

- **SGD has trouble navigating ravines.**

In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

- **Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations.**
- **SGD Momentum is motivated from the concept of momentum in physics.**
 - The optimization process resembles a heavy ball rolling down the hill. Momentum keeps the ball moving in the same direction that it is already moving in.
 - Gradient can be thought of as a force pushing the ball in some other direction.

CNN-SGD with Momentum. Formulars



Fig.1 SGD without momentum



Fig.2 SGD with momentum

- In Momentum, we have two iterates (m and w) instead of just one. The updates are as follows:

$$m_{k+1} = \beta_k m_k - \nabla l_i(w_k)$$

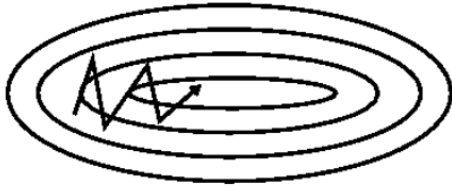
$$w_{k+1} = w_k - \eta_k m_{k+1}$$

where m is SGD momentum.

1. add the stochastic gradient to the old value of the momentum, after dampening it by a factor β ($[0,1]$).
2. we move w in the direction of the new momentum m .

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions

CNN-SGD with Momentum. Practical guidelines



SGD with momentum

- In Momentum, we have two iterates (m and w) instead of just one. The updates are as follows:

$$m_{k+1} = \beta_k m_k - \nabla l_i(w_k)$$

$$w_{k+1} = w_k - \eta_k m_{k+1}$$

- $\beta = 0.9$ or 0.99 are empirically selected as appropriate values in the SGD with Momentum .
- η_k learning rate usually needs to be decreased when the momentum parameter is increased to maintain convergence. e.g. **If β changes from 0.9 to 0.99 , learning rate must be decreased by a factor of 10.**

CNN-Optimization with **Adaptive methods**

- SGD with momentum is the state-of-the-art optimization method for a lot of learning problems.
- In the SGD formulation, every single weight in network is updated using an equation with the same learning rate. (global η).
- The learning rate may work well for the beginning of the network may not work well for the latter sections of the network. This means **adaptive learning rates** by layer could be useful.
- The adaptive methods were designed to take the information gained from each weight into account in the optimization by adapting a learning rate for each weight individually.
- Root Mean Square Propagation (**RMSprop**) and Adaptive Moment Estimation (**ADAM**)

CNN-Optimization with Adaptive methods. **RMSprop**

$$v_{k+1} = \alpha_k v_k + (1 - \alpha) \nabla l_i(w_k)^2$$

$$w_{k+1} = w_k - \eta \frac{\nabla l_i(w_k)}{\sqrt{v_{k+1} + \epsilon}}$$

where η is the global learning rate, ϵ is a value close to machine ϵ (on the order of 10^{-8}) – in order to avoid division by zero errors, and v_{k+1} is the 2nd moment estimate.

Tricks

Hinton suggests α to be set to 0.9, while a good default value for the learning rate η is 0.001.

- Root Mean Square Propagation (**RMSprop**) was introduced by Geoff Hinton. (The 2018 Turing Award)
- The gradient is normalized by its root-mean-square.
- We need to put larger weights on the newer values as they provide more information.
- One way to do that is down-weight old values exponentially. The values in the v calculation that are very old are down-weighted at each step by an α [0,1] constant.
- We update v to estimate this quantity via an *exponential moving average* and The second moment is used to normalize the gradient element-wise

CNN-Optimization with Adaptive methods. (ADAM)

Adaptive Moment Estimation (ADAM)

$$\mathbf{m}_{k+1} = \beta \mathbf{m}_k + (1 - \beta) \nabla l_i(\mathbf{w}_k)$$

$$\mathbf{v}_{k+1} = \alpha \mathbf{v}_k + (1 - \alpha) \nabla l_i(\mathbf{w}_k)^2$$

$$\hat{\mathbf{m}}_{k+1} = \mathbf{m}_{k+1} / (1 - \beta^k)$$

$$\hat{\mathbf{v}}_{k+1} = \mathbf{v}_{k+1} / (1 - \alpha^k)$$

Next the first and second moments are bias-corrected

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{\hat{\mathbf{m}}_{k+1}}{\sqrt{\hat{\mathbf{v}}_{k+1} + \epsilon}}$$

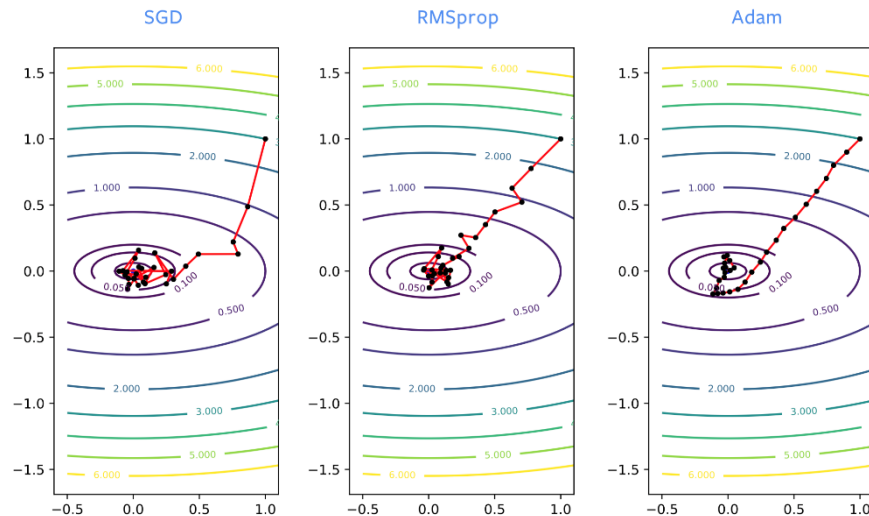
where η is the global learning rate, ϵ is a value close to machine ϵ (on the order of 10^{-8}) – in order to avoid division by zero errors, and \mathbf{v}_{k+1} is the 2nd moment estimate.

Tricks

The authors propose default values of 0.9 for β , 0.999 for α and 10^{-8} for ϵ . They show empirically that Adam works well in practice

- Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.
- ADAM is a more commonly used method, which is RMSprop plus momentum,
- storing an exponentially decaying average of past squared gradients v like RMSprop
- It also keeps an exponentially decaying average of past gradients momentum m .

CNN-Optimization. (SGD vs RMSprop vs ADAM)



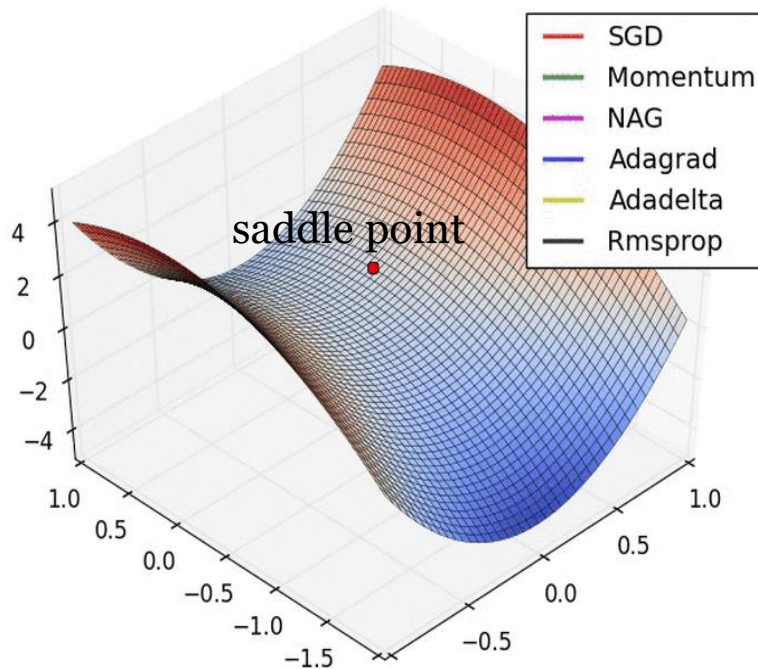
SGD vs. RMSprop vs. ADAM²

- *SGD often goes in the wrong direction in the beginning of the training process, whereas RMSprop hones in on the right direction.*
- *RMSprop suffers from noise just as regular SGD, so it bounces around the optimum significantly once it's close to a local minimizer.*
- *Just like when we add momentum to SGD, we get the same kind of improvement with ADAM.*
- *ADAM is generally recommended over RMSprop*

1

1. <https://atcold.github.io/pytorch-Deep-Learning/en/week05/05-2/>
2. https://atcold.github.io/pytorch-Deep-Learning/images/week05/05-2/5_2_comparison.png

CNN-Optimization Algorithms. Visualization



Nesterov accelerated gradient (NAG) ¹, Adagrad ², Adadelta ³, AdamW ⁴, QHAdam ⁵, AggMo ⁶

- The animation provides some intuitions towards the optimization behaviour of some well-known optimization methods, at a saddle point
- the adaptive learning-rate methods, are most suitable and provide the best convergence for these scenarios.

¹Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $\mathcal{O}(1/k^2)$. Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543–547.

²Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.

³Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from <http://arxiv.org/abs/1212.5701>

Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. In Proceedings of ICLR 2019.

⁴Ma, J., & Yarats, D. (2019). Quasi-hyperbolic momentum and Adam for deep learning. In Proceedings of ICLR 2019.

⁵Lucas, J., Sun, S., Zemel, R., & Grosse, R. (2019). Aggregated Momentum: Stability Through Passive Damping. In Proceedings of ICLR 2019. <https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants>

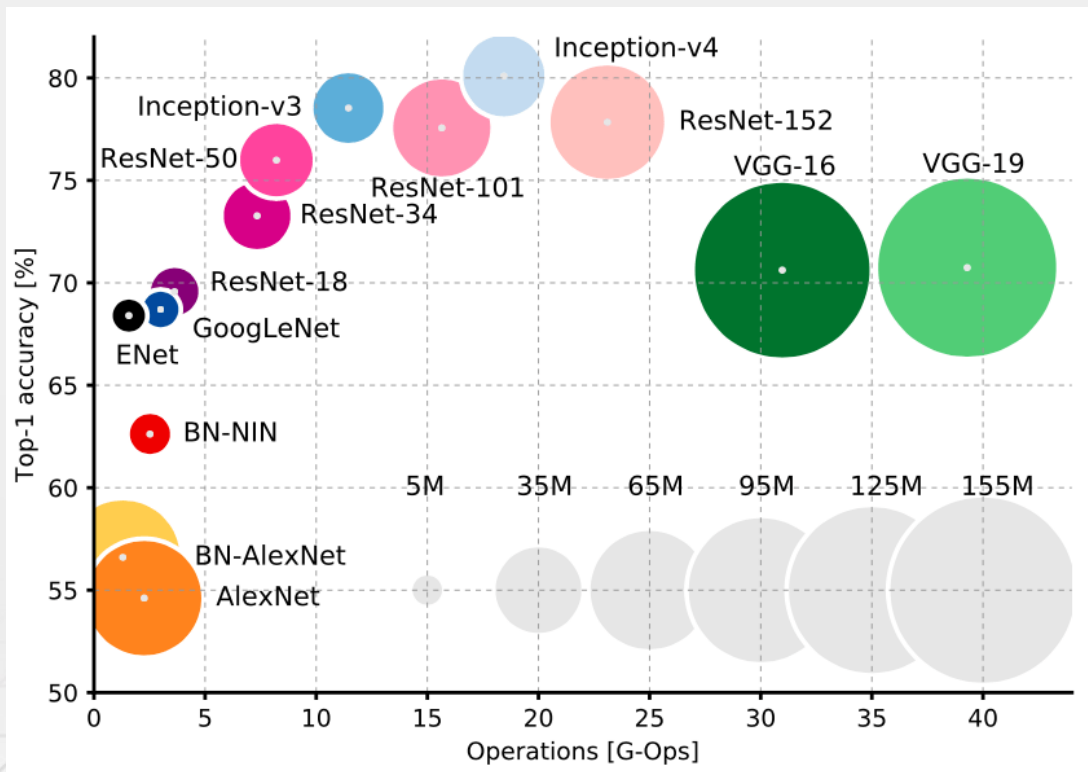


Outline

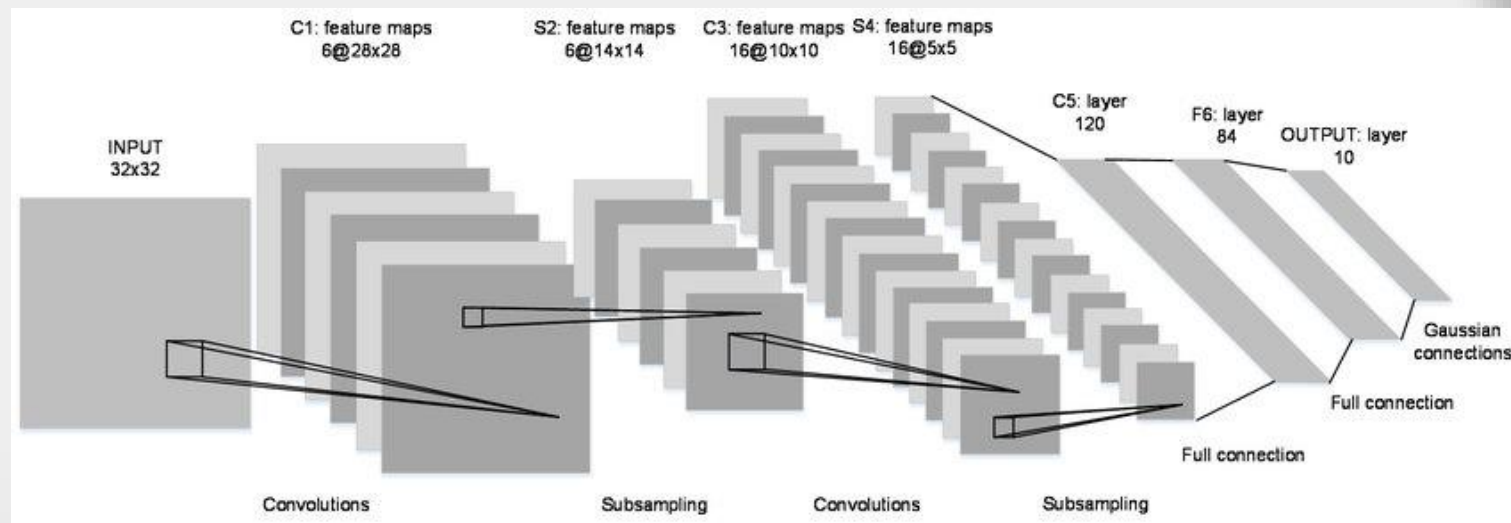
Covering:

- **Softmax + cross entropy loss for classification**
(softmax activation, Softmax with cross entropy loss, image classification in practice)
- **Optimization methods**
(Gradient descent, Learning rate, Full Gradient descent, Stochastic Gradient Descent (SGD), Momentum and adaptive method)
- **Classic CNN architectures**
(LeNet, AlexNet, Zfnet, VGG, ResNet, and their evolution)

CNN architecture



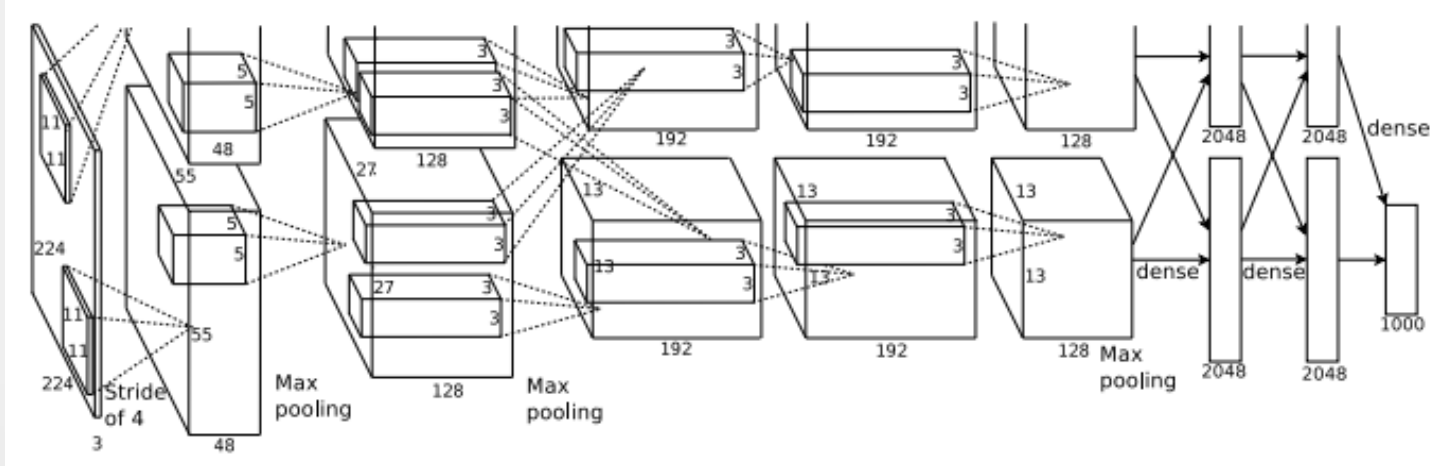
CNN architecture-LeNet in 1998



- This (LeNet) is one of the very first implementations of CNN
- It is a **7 blocks** convolutional network that classifies digits and used by several banks to recognise hand-written numbers on cheques digitized in 32x32 pixel greyscale input images.
- This led to a research into deep learning and gave rise to ground breaking models.

LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). Proceedings of the IEEE. 86 (11): 2278–2324. doi:10.1109/5.726791

CNN architecture-AlexNet in 2012



- It won the ImageNet challenge in 2012.
- AlexNet is considered to be the first CNN architecture which rose the interest in CNNs.
- The net contains **8 layers** with weights; the first 5 are convolutional and the remaining 3 are fullyconnected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

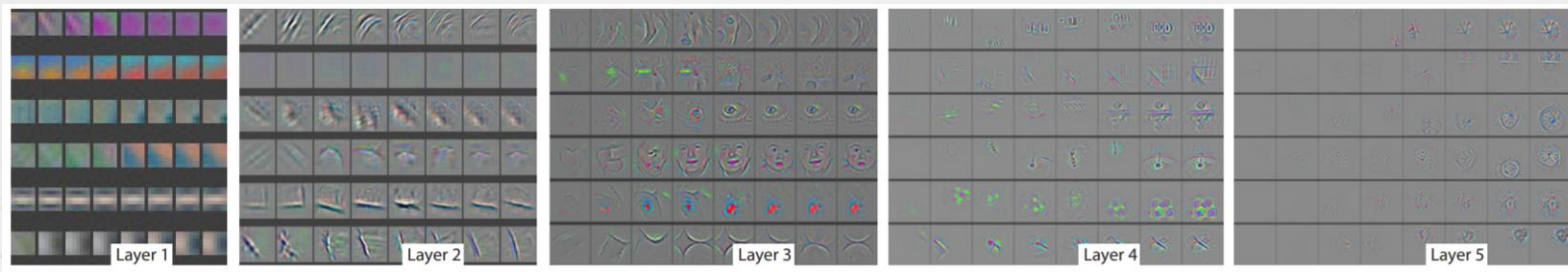
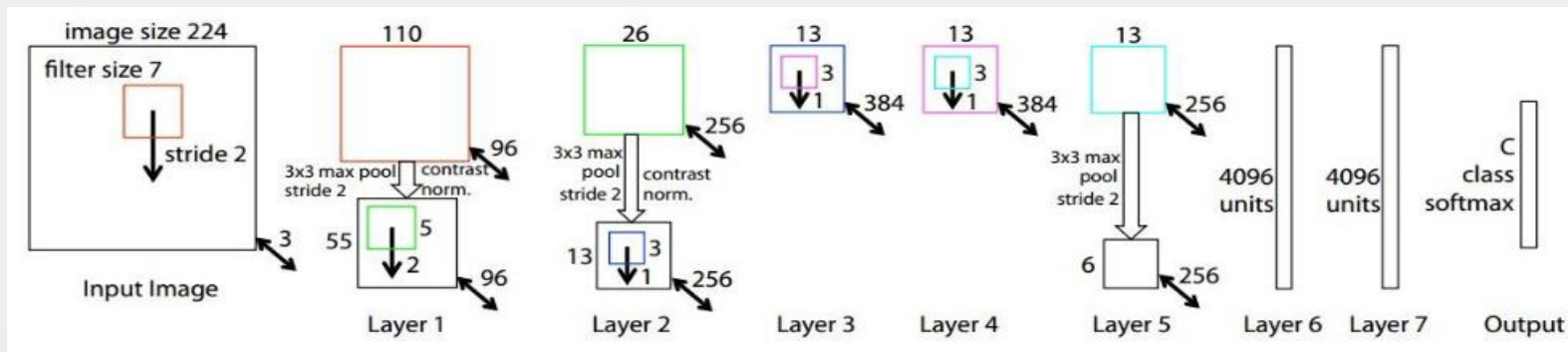


There is no clear understanding of why they perform so well, or how they might be improved.

CNN architecture-ZFNet visualization

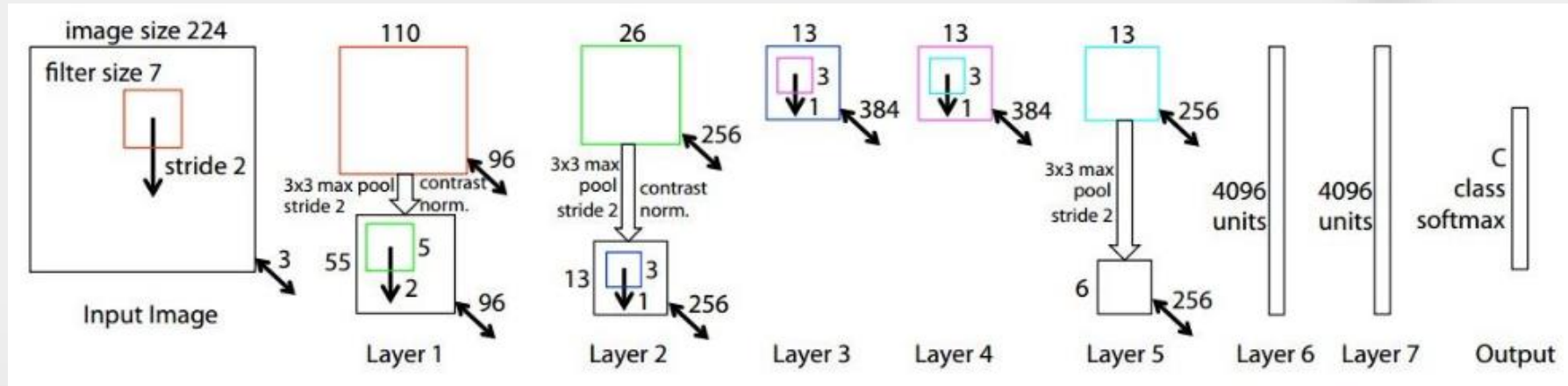


They introduced a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier.



Evolution of a randomly chosen subset of model features through training. Each layer's features are displayed in a different block.

CNN architecture-ZFNet in 2013



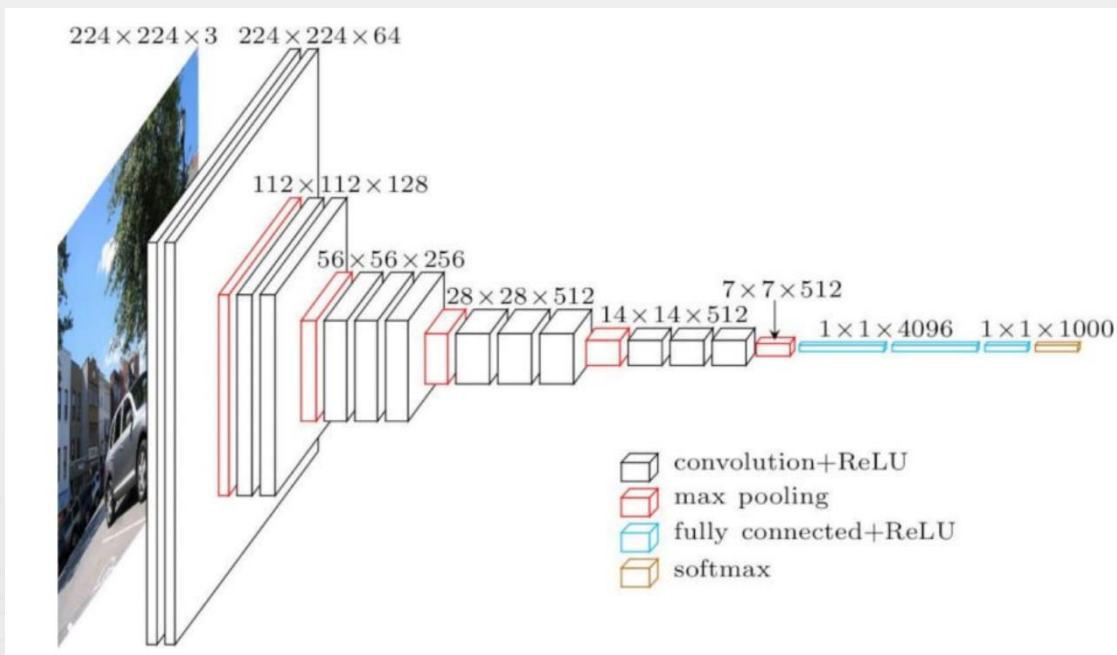
- One major difference in the approaches was that ZF Net used 7x7 sized filters whereas AlexNet used 11x11 filters. ZF Net utilised smaller receptive window size and smaller stride of the first convolutional layer
- The intuition behind this is that by using bigger filters we were losing a lot of pixel information, which we can retain by having smaller filter sizes in the earlier conv layers.
- This network also used ReLUs for activation and trained using batch stochastic gradient descent.

CNN architecture-VGG16 2014



In addition to the filter kernel size, Oxford team addressed another important aspect of ConvNet architecture design – its depth.

CNN architecture-VGG16 2014



The main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to **16–19** weight layers

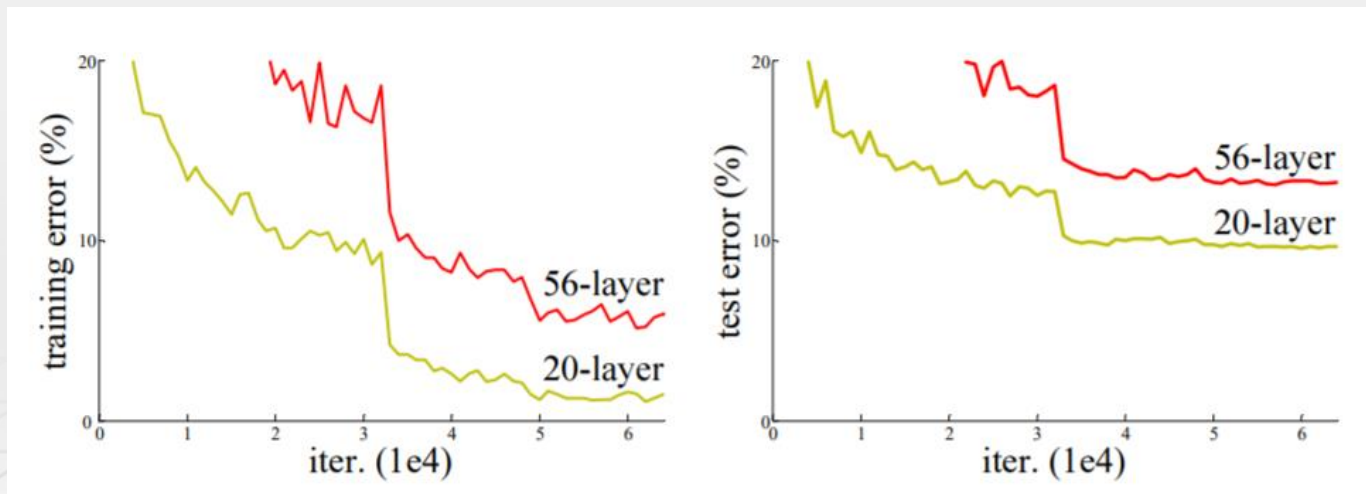


Driven by the significance of depth, a question arises: Is learning better networks as easy as stacking more layers?

CNN architecture-The problem of very deep neural networks



- One of the major benefits of a very deep network is that it can represent very complex functions.
- However, a huge barrier to training them is vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent prohibitively slow.



Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error.

CNN architecture-The problem of vanishing/exploding gradient



More specifically, during gradient descent, as we backprop from the final layer back to the first layer, we are multiplying by the weight matrix on each step. If the gradients are small, due to large number of multiplications, the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and “explode” to take very large values).

- $0.9^2 = 0.81$

- $0.9^{10} = 0.35$

-

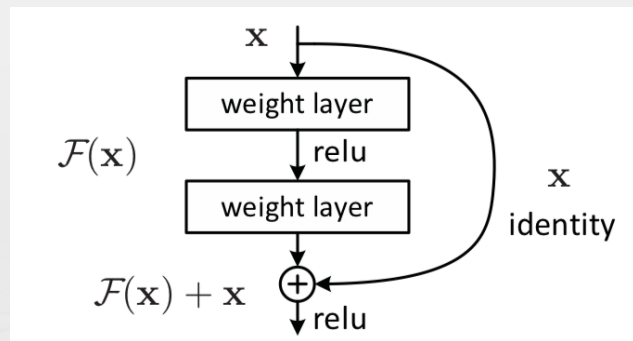
- $0.9^{100} = ?$

$0.9^{100} = 0.000026$

CNN architecture-ResNet in 2015



- They addressed the degradation problem by introducing a deep residual learning framework.
- Instead of hoping each few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping.



the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers (Fig. 1).

Fig1. Residual learning: a building block

CNN architecture-VGG16 and ResNet

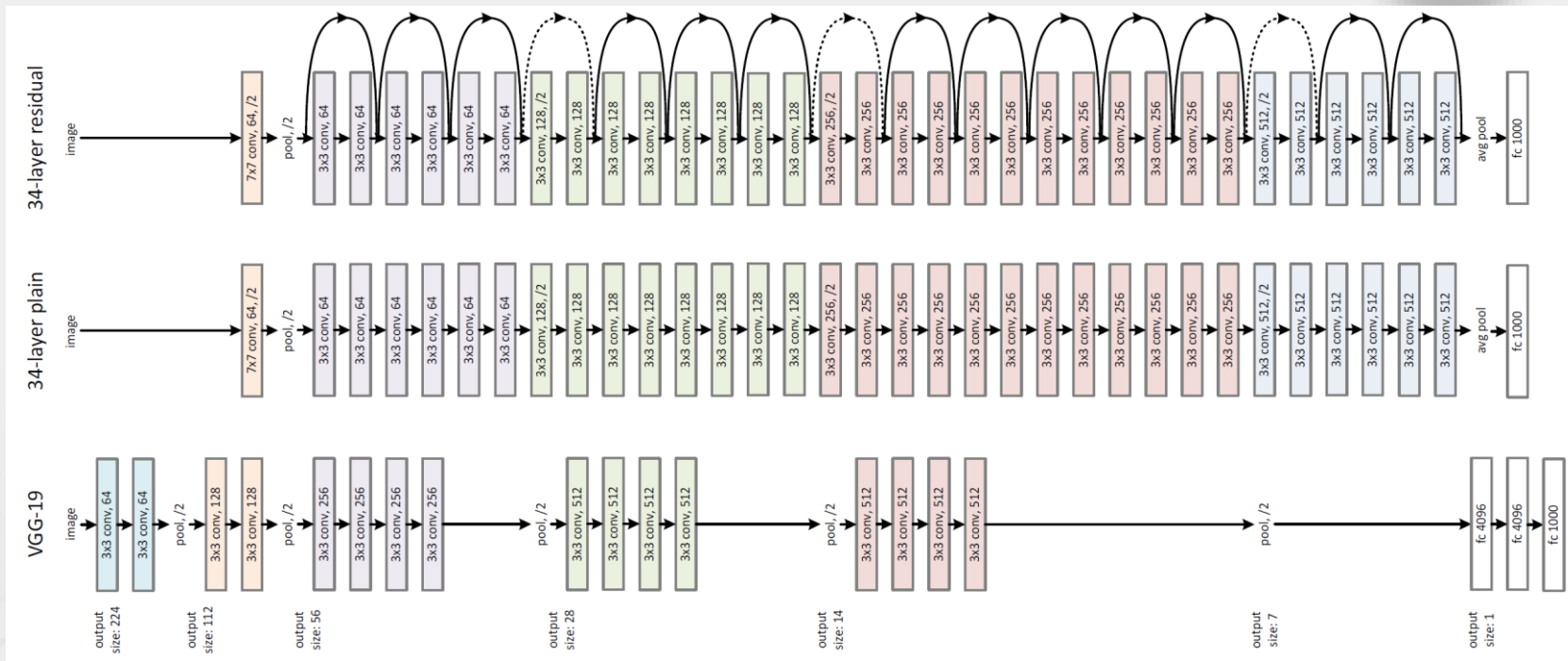


Figure 1. Example network architectures for ImageNet. Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions.

CNN architecture-ResNet

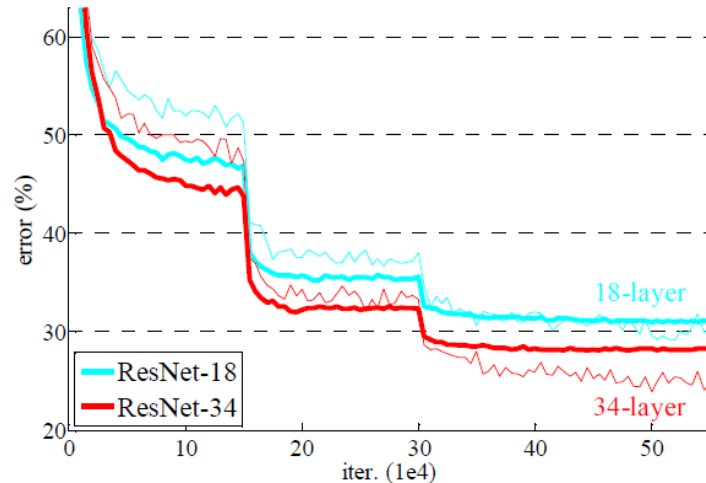
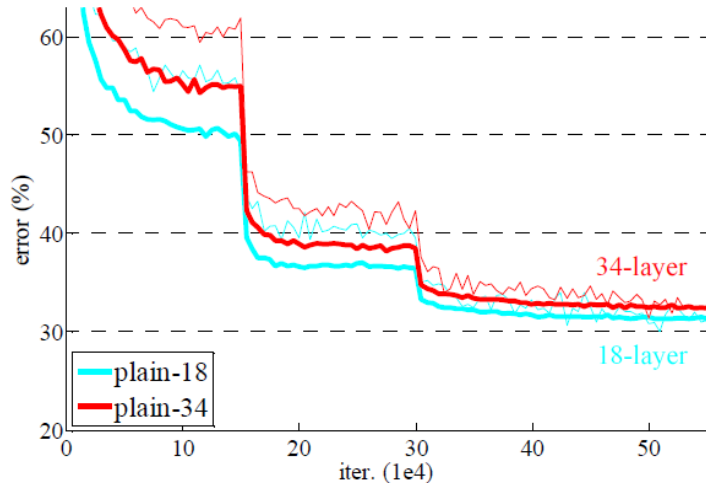


Figure 4. Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

the degradation problem - 34-layer plain net has higher training error throughout the whole training procedure

CNN architecture-YourNet in future



?



Summary

- **Softmax + cross entropy loss for classification**
(softmax activation, Softmax with cross entropy loss, image classification in practice)
- **Optimization methods**
(Gradient descent, Learning rate, Full Gradient descent, Stochastic Gradient Descent (SGD), Momentum and adaptive method)
- **Classic CNN architectures**
(LeNet, AlexNet, Zfnet, VGG, ResNet, and their evolution)



Thank you!

Reference:

- LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition" (PDF). *Proceedings of the IEEE*. 86 (11): 2278–2324. doi:10.1109/5.726791
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *NIPS*, pp. 1106–1114, 2012.
- Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. Published in *Proc. ECCV*, 2014.
- K. Simonyan and A. Zisserman. (2015) Very deep convolutional networks for large-scale image recognition. In *ICLR*.
- K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.
- Navarro, C.A., Hitschfeld-Kahler, N. and Mateu, L., 2014. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Communications in Computational Physics*, 15(2), pp.285–329.
- Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations*, 1–13.
- Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady ANSSSR* (translated as *Soviet.Math.Docl.*), vol. 269, pp. 543– 547.
- Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12, 2121–2159.
- Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Retrieved from <http://arxiv.org/abs/1212.5701>
- Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization. In *Proceedings of ICLR 2019*.
- Ma, J., & Yarats, D. (2019). Quasi-hyperbolic momentum and Adam for deep learning. In *Proceedings of ICLR 2019*.
- Lucas, J., Sun, S., Zemel, R., & Grosse, R. (2019). Aggregated Momentum: Stability Through Passive Damping. In *Proceedings of ICLR 2019*.