CS 4602

Introduction to Machine Learning

Deep learning

Instructor: Po-Chih Kuo

Roadmap

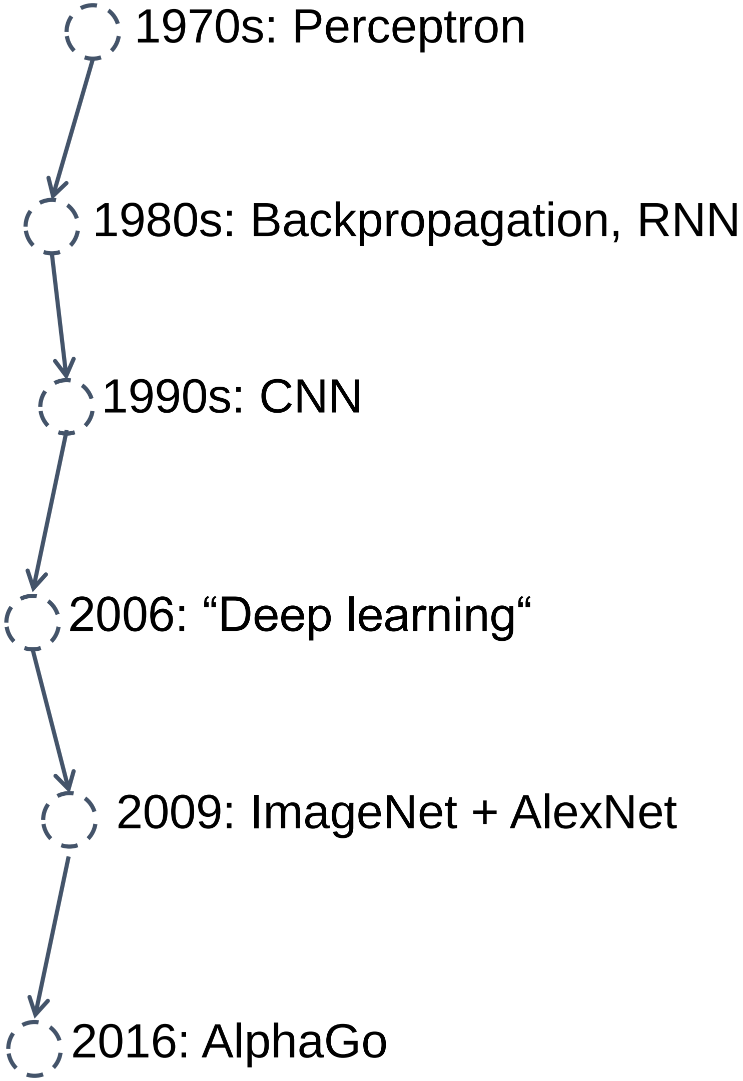
- Introduction and Basic Concepts
- Regression
- Bayesian Classifiers
- Decision Trees
- Linear Classifier
- Neural Networks
- Deep learning
- Convolutional Neural Networks
- Reinforcement Learning
- KNN
- Model Selection and Evaluation
- Clustering
- Data Exploration & Dimensionality reduction

Outline

- Why deep?
- Optimization
- Regularization (Overfitting problem)

Tricks!

Why Deep Learning?



1970s: Perceptron

1980s: Backpropagation, RNN

1990s: CNN

2006: “Deep learning“

2009: ImageNet + AlexNet

2016: AlphaGo

Deep learning:

- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

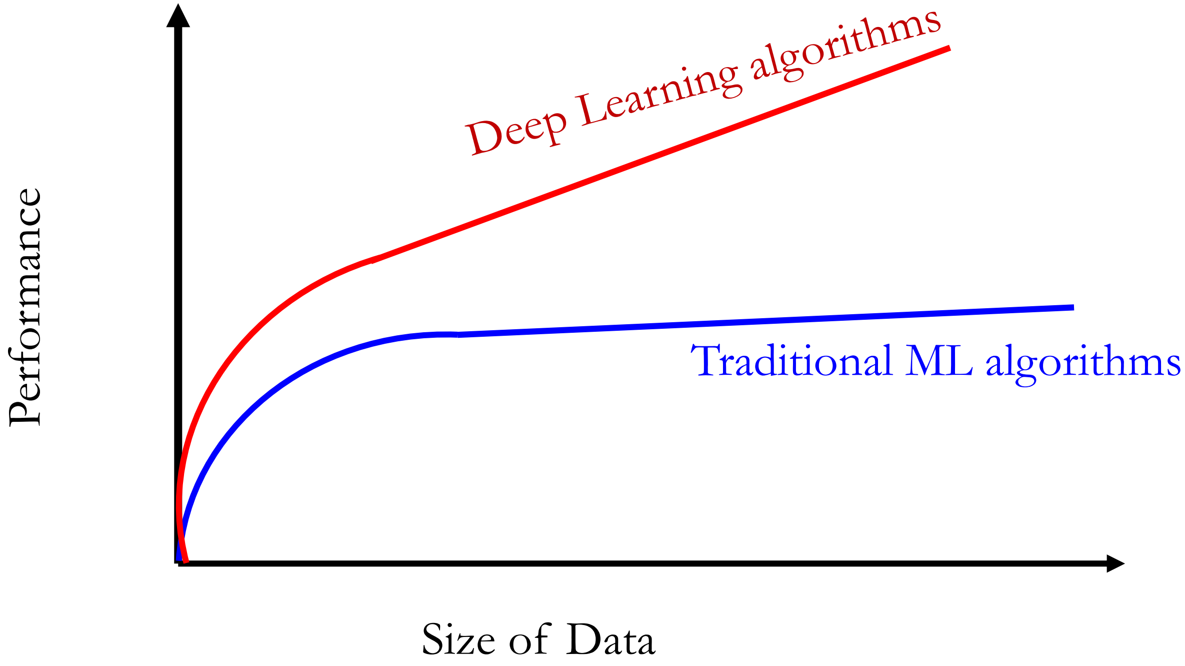
This wasn’t always the case!

Since 1980s: Form of models hasn’t changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)

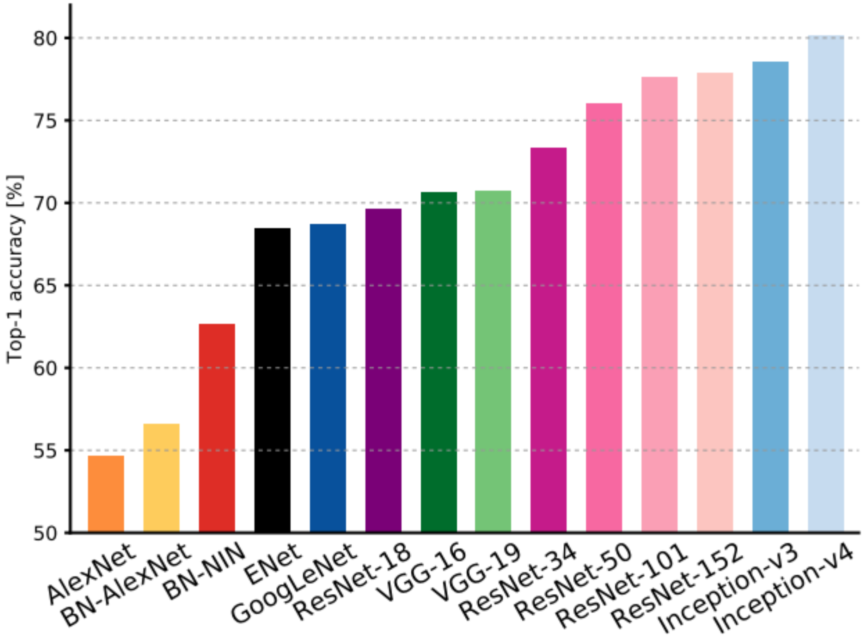
Ref: Y. LeCun DL lecture slides

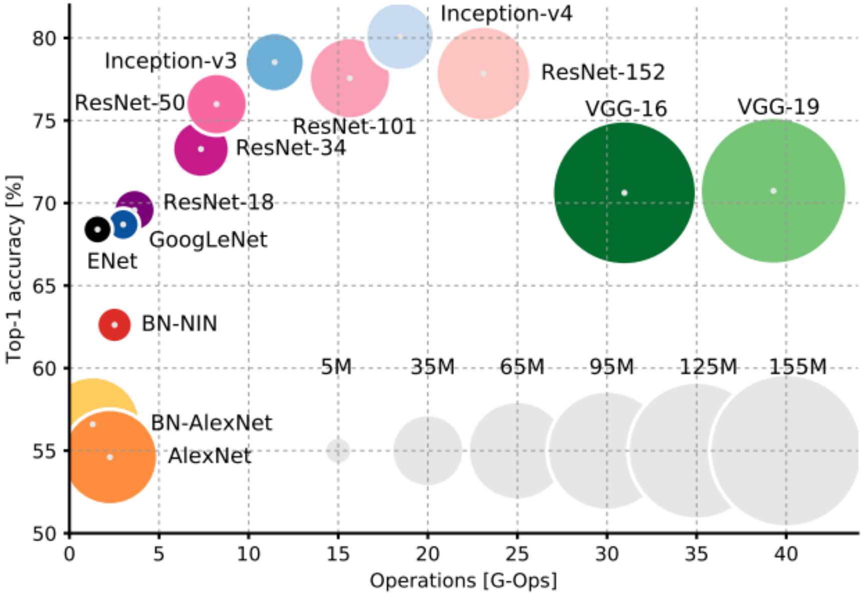
Performance vs Sample Size



Parameters vs Accuracy on ImageNet

(Canziani 2016)





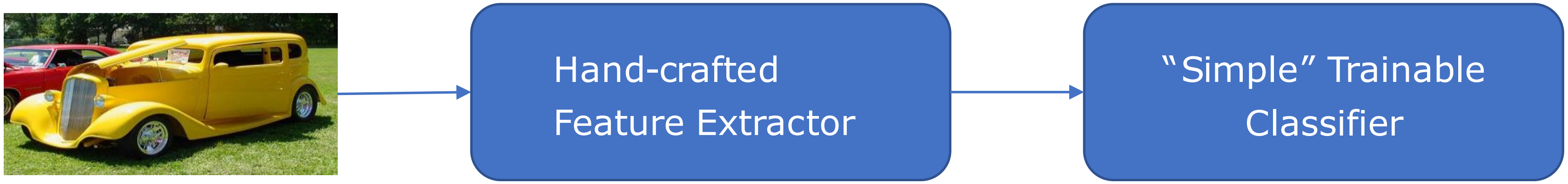
Giga Operations Per Second

- ResNet50 and ResNet100 are used routinely in production.
- Each of the few billions photos uploaded on Facebook every day goes through a handful of ConvNets within 2 seconds.

Ref: Y. LeCun DL lecture slides

Machine Learning = Learning Representations

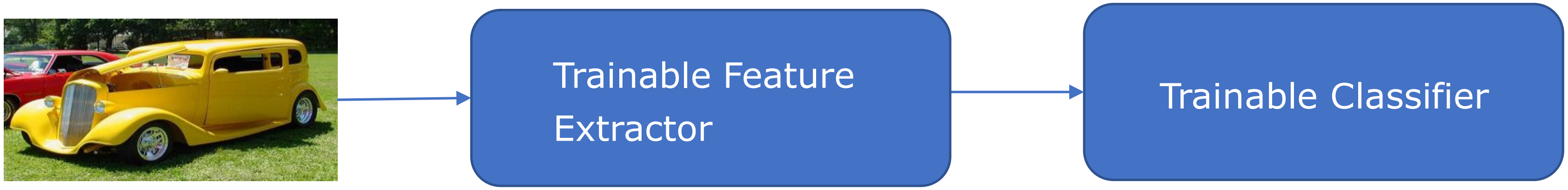
- **The traditional model of pattern recognition (since the late 50's)**
  - Fixed/engineered features (or fixed kernel) + trainable classifier



Hand-crafted Feature Extractor

“Simple” Trainable Classifier

- **End-to-end learning / Feature learning / Deep learning**
  - Trainable features (or kernel) + trainable classifier

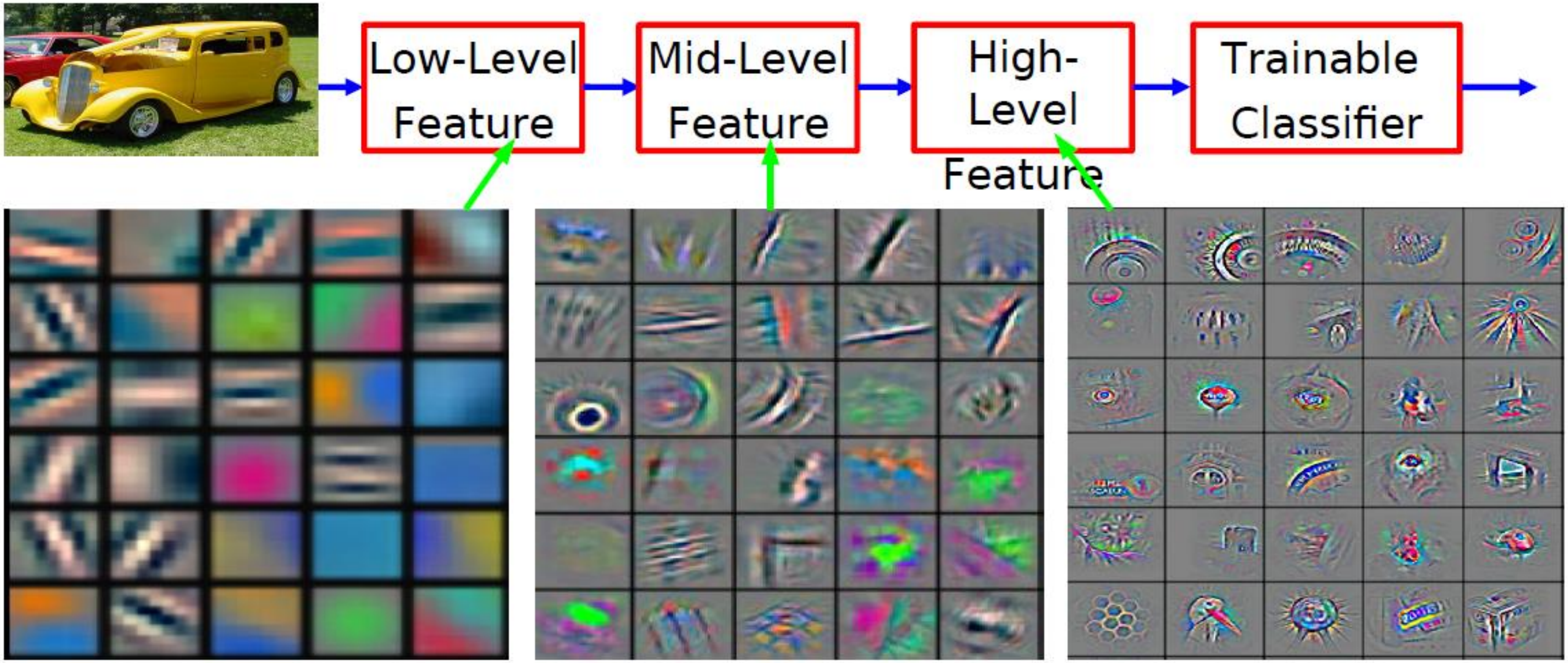


Trainable Feature Extractor

Trainable Classifier

Ref: Y. LeCun DL lecture slides

Deep Learning = Learning Hierarchical Representations



Low-Level Feature

Mid-Level Feature

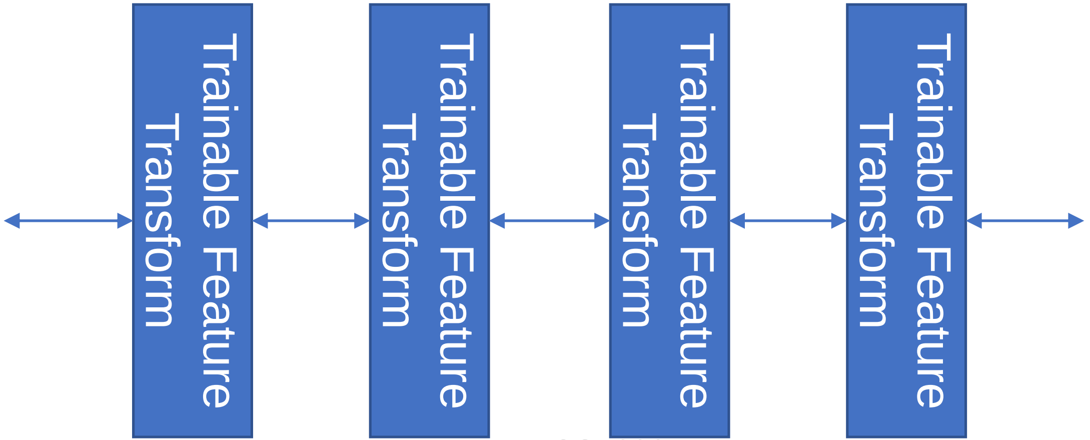
High-Level Feature

Trainable Classifier

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Hierarchical representation

- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word


Trainable Feature Transform
Trainable Feature Transform
Trainable Feature Transform
Trainable Feature Transform

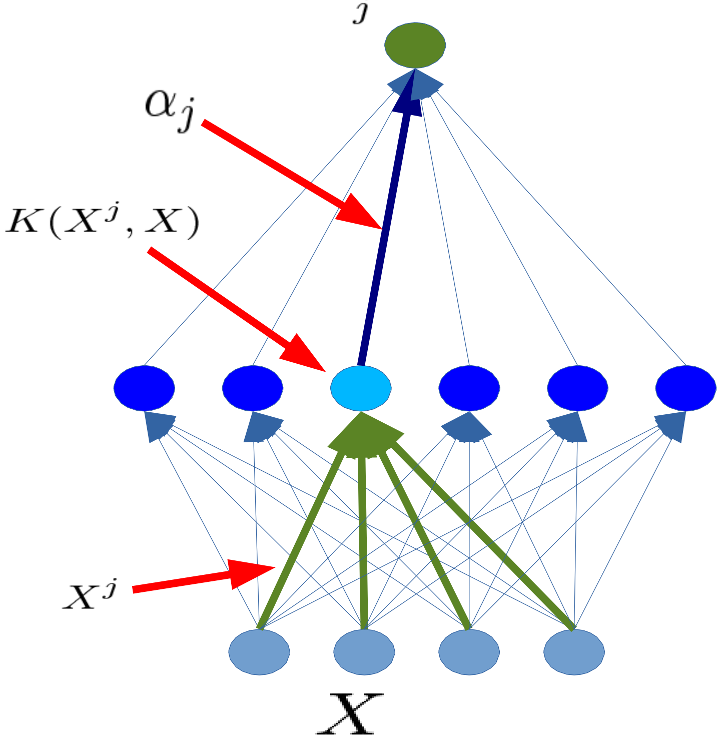
Ref: Y. LeCun DL lecture slides

Which Models are Deep?

MLP, SVM, or DT

- 2-layer perceptrons are not deep
  - Because there is no feature hierarchy
- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
  - The first layer is “trained” using the samples as templates for the kernel functions.
- Decision trees are not deep
  - No hierarchy of features. All decisions are made in the input space

$$G(X,\alpha) = \sum_j \alpha_j K(X^j, X)$$



Ref: Y. LeCun DL lecture slides

Outline

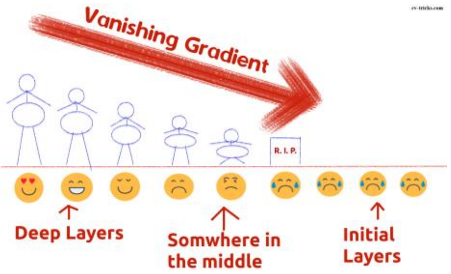
- Why deep?
- Optimization
- Regularization (Overfitting problem)

Optimization

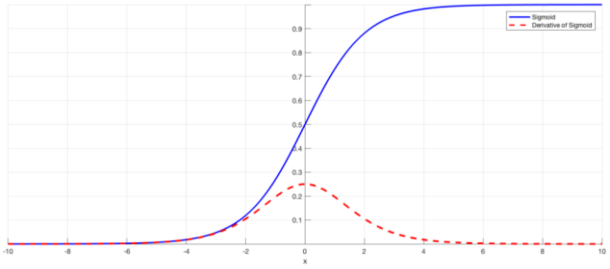
- Vanishing Gradient Problem
- Stochastic Gradient Descent (SGD)
- Momentum Method
- Adaptive Learning Methods (AdaGrad, RMSProp, Adam)

Vanishing Gradient Problem

- Back propagation uses chain rule.
  - Chain rule multiplies derivatives
- If these derivatives are between 0 and 1 the product vanishes as the chain gets longer.
  - or the product explodes if the derivatives are greater than 1.
- Sigmoid activation function leads to this problem.
- Saturating: After some epochs that learning happens relatively fast, the value saturates, and it takes much time to update the weights, because the value of gradient is small.



Sigmoid

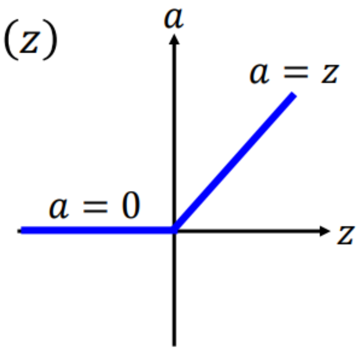


Relu (Rectified Linear Units) function

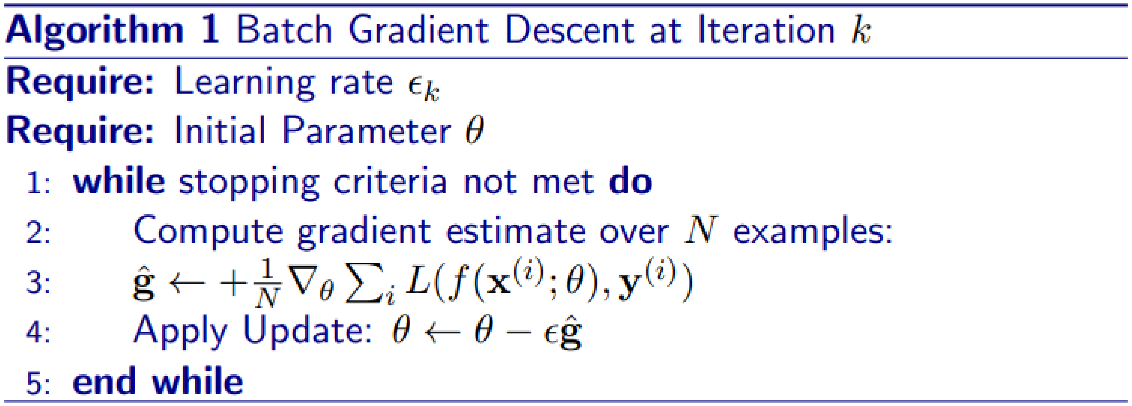
- Avoids Vanishing Gradient Problem
  - Fast to compute
  - Biological reason
- The formula is simple: max(0,z).

$$\frac{d}{dx}ReLU(x) = \begin{cases} 0, & \text{if } x < 0, \\ 1, & \text{otherwise.} \end{cases}$$

ReLU(z)

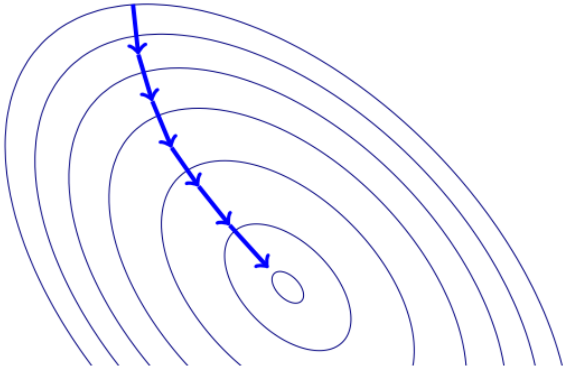


Batch Gradient Descent



Algorithm 1 Batch Gradient Descent at Iteration k

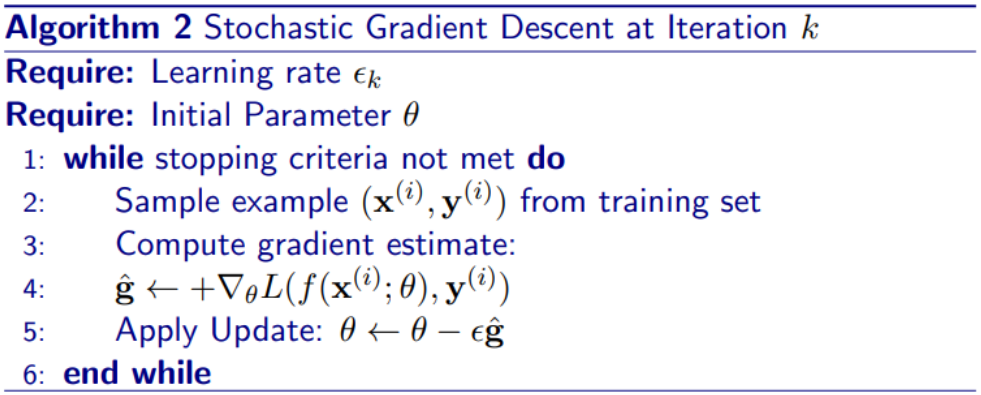
```
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
while stopping criteria not met do
    Compute gradient estimate over $N$ examples:
    $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while
```



- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

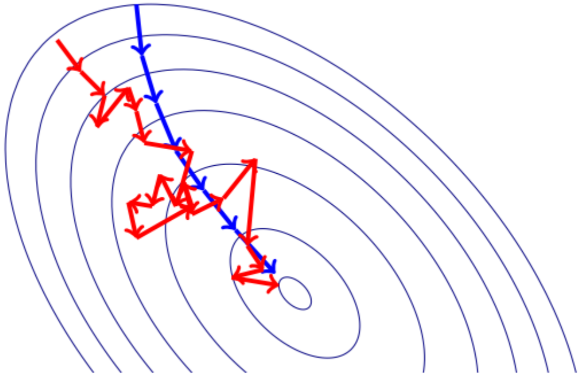
Ref: G0 Hinton lecture slides

Stochastic Gradient Descent



Algorithm 2 Stochastic Gradient Descent at Iteration k

```
Require: Learning rate $\epsilon_k$
Require: Initial Parameter $\theta$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
end while
```



ϵ_k is learning rate at step k

In practice the learning rate is decayed linearly till iteration τ

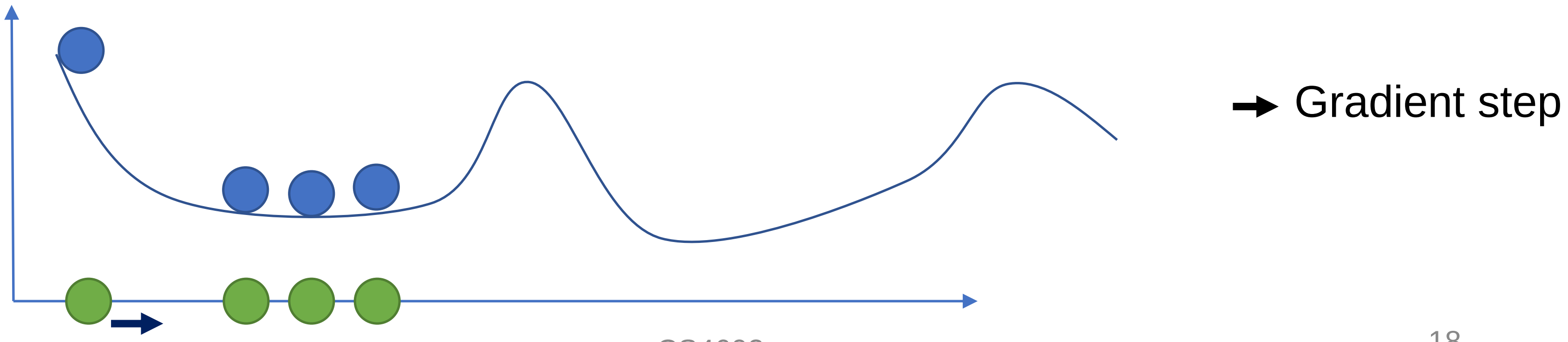
$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau \text{ with } \alpha = \frac{k}{\tau}$$

Minibatching

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches
- Advantage: Computation time per update does not depend on number of training examples N
- This allows convergence on extremely large datasets

Momentum

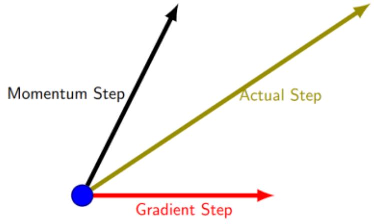
- The Momentum method is a method to accelerate learning using SGD
- In particular SGD suffers in the following scenarios:
    - Loss surface has high curvature
    - We get small but consistent gradients
    - The gradients are very noisy



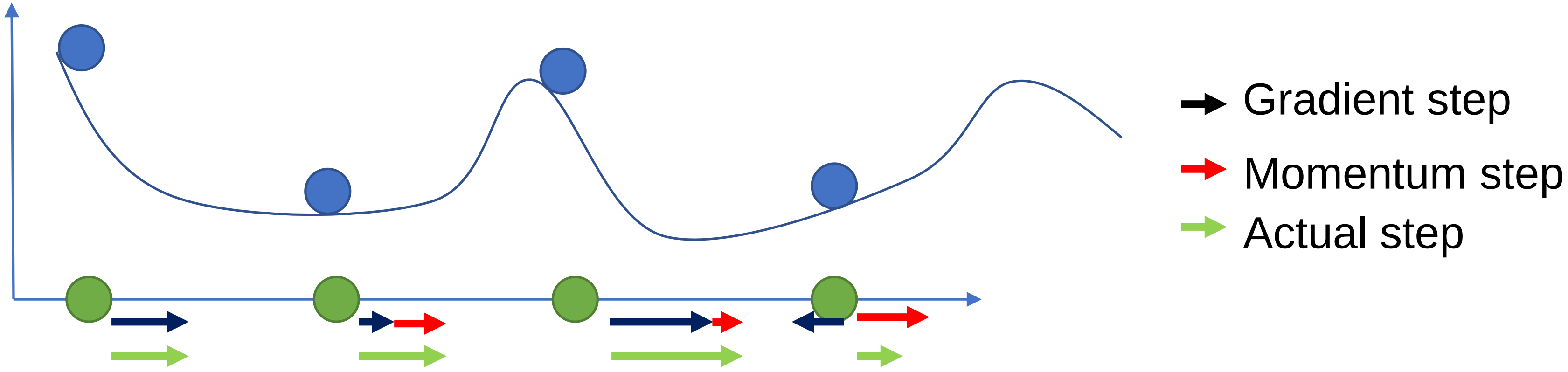
Momentum

- How do we try and solve this problem?
- Introduce a new variable v, the velocity
- We think of v as the direction and speed by which the parameters move as the learning dynamics progresses
- The velocity is an exponentially decaying moving average of the negative gradients

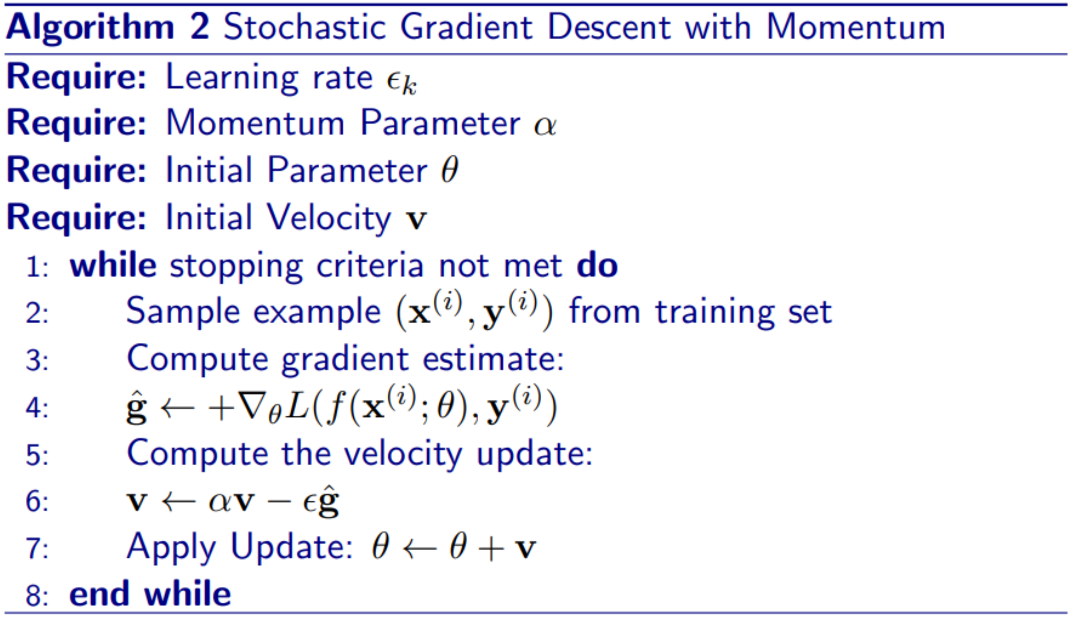
$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$



- Update rule: $\theta \leftarrow \theta + \mathbf{v}$
- Usually values for α are set high ≈ 0.8, 0.9

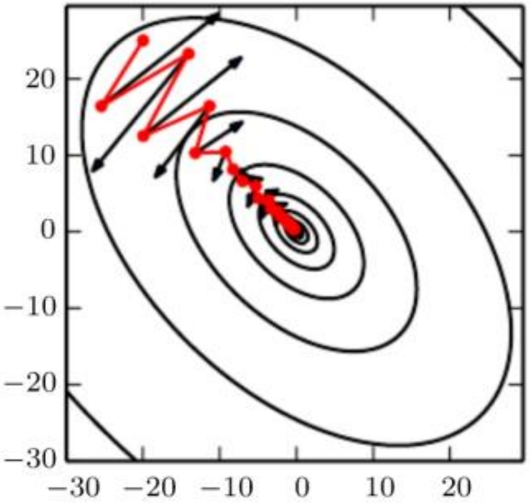

Gradient step
Momentum step
Actual step

SGD with Momentum



Algorithm 2 Stochastic Gradient Descent with Momentum

```
Require: Learning rate $\epsilon_k$
Require: Momentum Parameter $\alpha$
Require: Initial Parameter $\theta$
Require: Initial Velocity $\mathbf{v}$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Compute the velocity update:
    $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \mathbf{v}$
end while
```



Nesterov Momentum

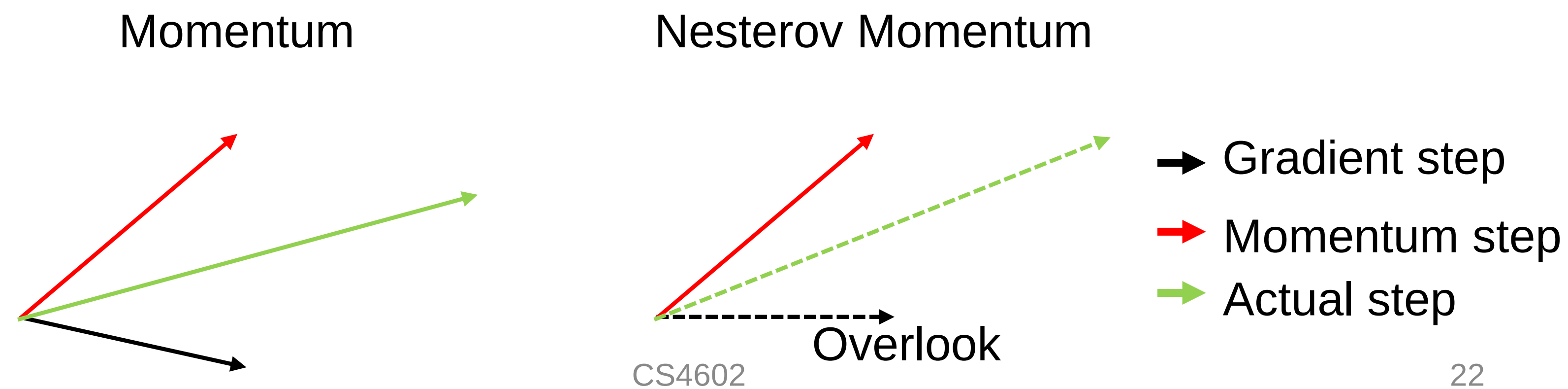
- The velocity term in the Momentum method:

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\nabla_\theta\left(L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})\right)$$

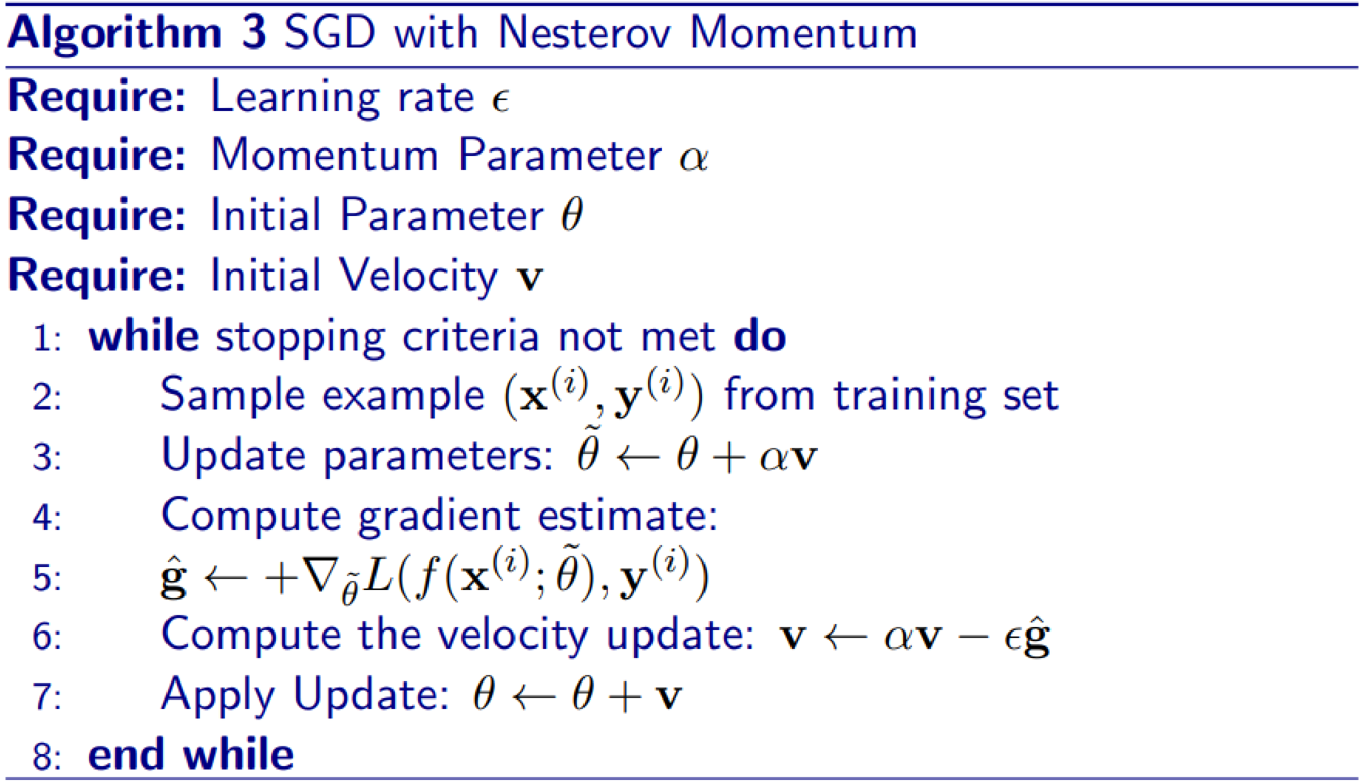
- Nesterov Momentum:

$$\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\nabla_\theta\left(L(f(\mathbf{x}^{(i)};\theta+\alpha\mathbf{v}),\mathbf{y}^{(i)})\right)$$

- Update: $\theta \leftarrow \theta + \mathbf{v}$



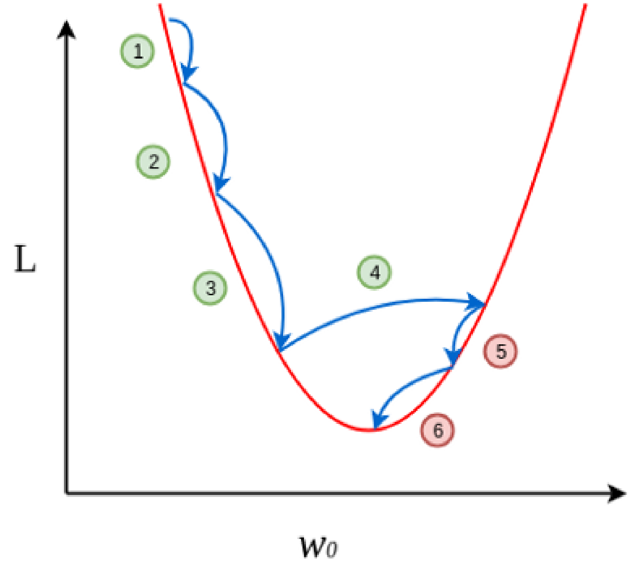
SGD with Nesterov Momentum =Nesterov Accelerated Gradient (NAG)



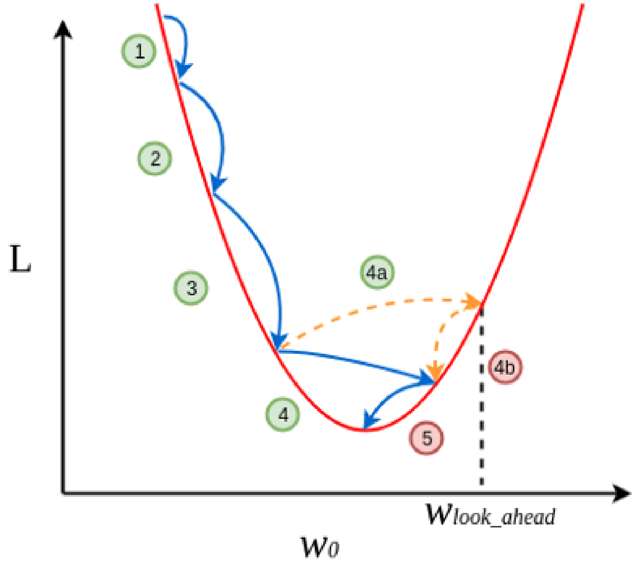
Algorithm 3 SGD with Nesterov Momentum

```
Require: Learning rate $\epsilon$
Require: Momentum Parameter $\alpha$
Require: Initial Parameter $\theta$
Require: Initial Velocity $\mathbf{v}$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Update parameters: $\tilde{\theta} \leftarrow \theta + \alpha\mathbf{v}$
    Compute gradient estimate:
    $\hat{\mathbf{g}} \leftarrow +\nabla_{\tilde{\theta}} L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$
    Compute the velocity update: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \epsilon\hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \mathbf{v}$
end while
```

Momentum SGD vs. NAG



(a) Momentum-Based Gradient Descent



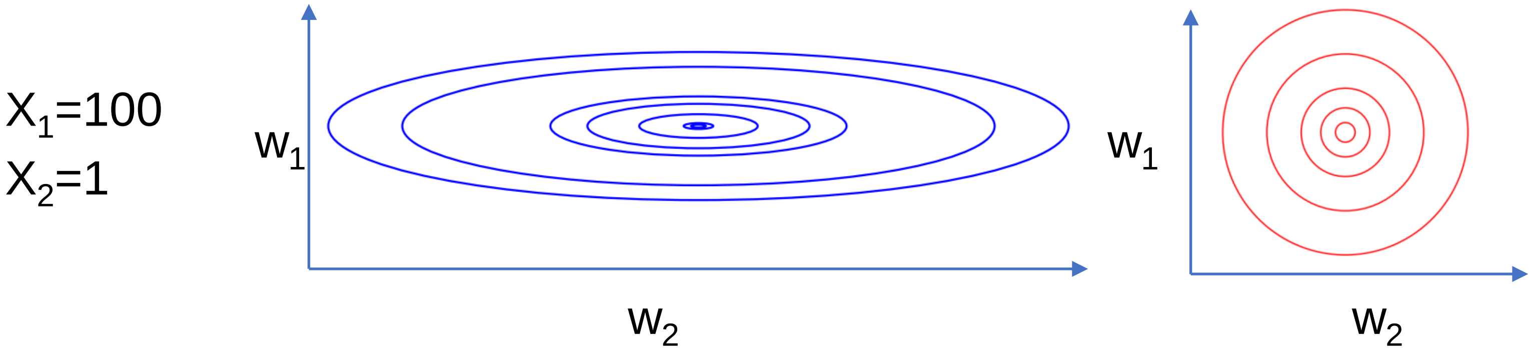
(b) Nesterov Accelerated Gradient Descent

🟢 $\Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$ 🔴 $\Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$

Ref: https://mc.ai/learning-parameters-part-2-momentum-based-and-nesterov-accelerated-gradient-descent/

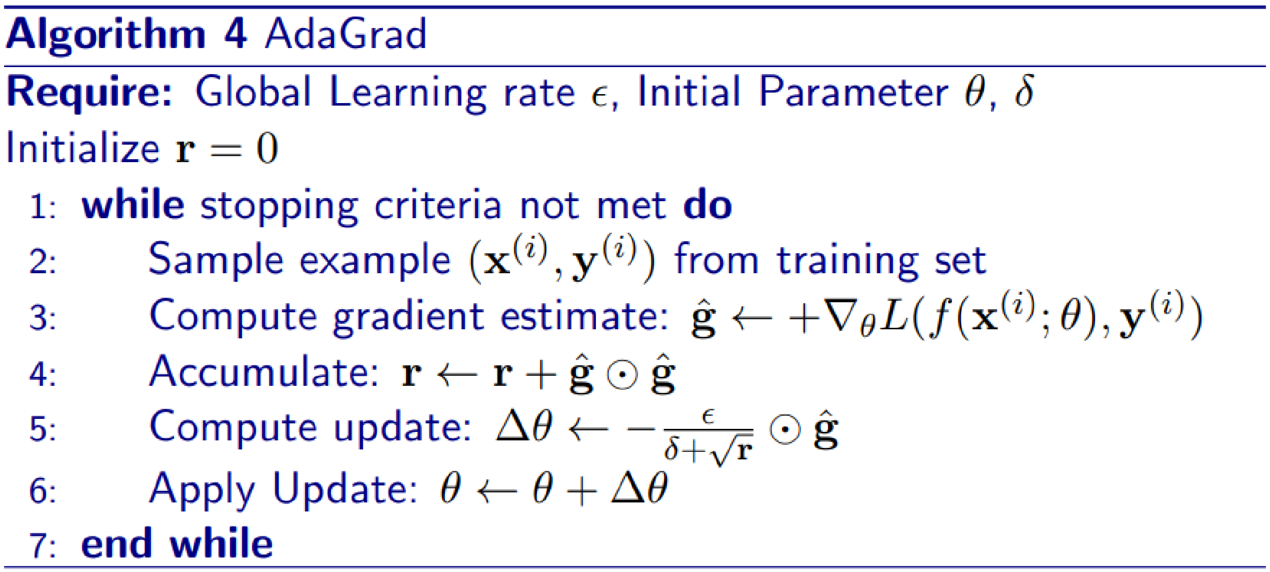
Adaptive Learning Rate Methods

- Motivation
  - Till now we assign the same learning rate to all features
  - If the features vary in importance and frequency, why is this a good idea? It’s probably not!



AdaGrad

- Idea: Downscale a model parameter by square-root of sum of squares of all its historical values
- Parameters that have large partial derivative of the loss → learning rates for them are rapidly declined



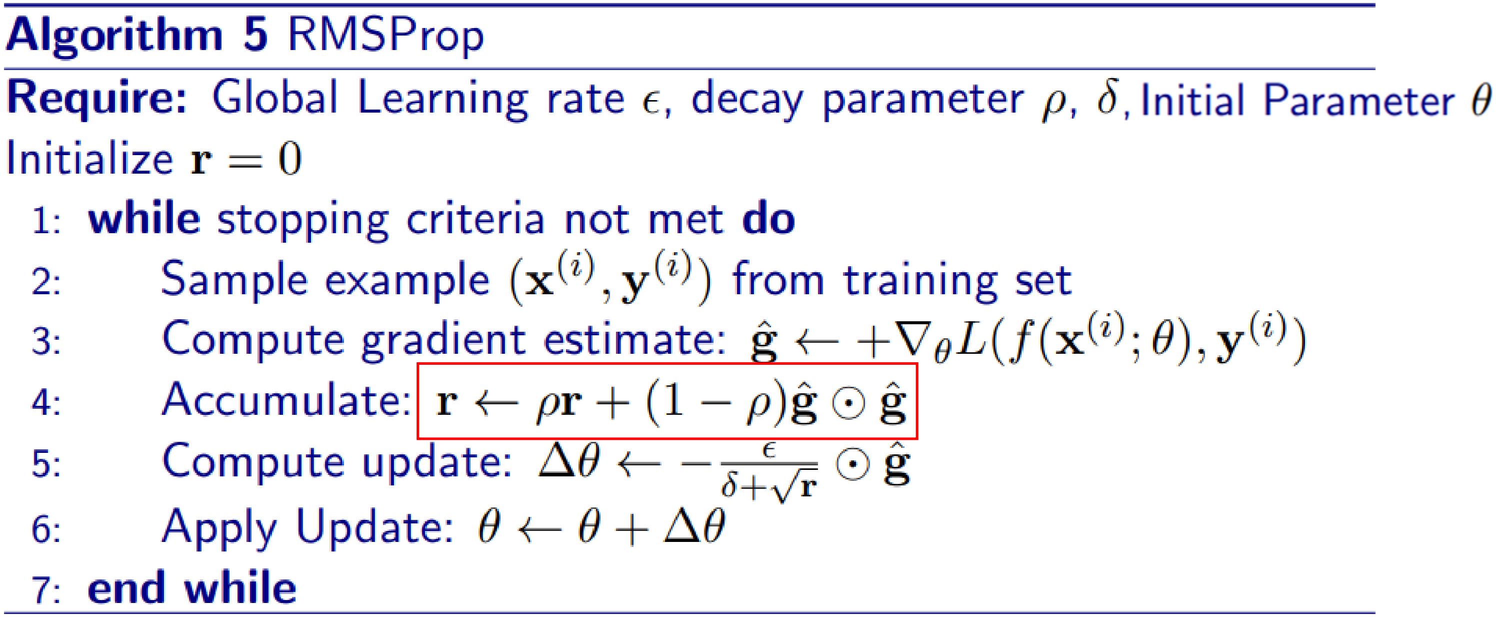
Algorithm 4 AdaGrad

```
Require: Global Learning rate $\epsilon$, Initial Parameter $\theta$, $\delta$
Initialize $\mathbf{r} = 0$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Accumulate: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```

RMSProp

Root Mean Squared Propagation

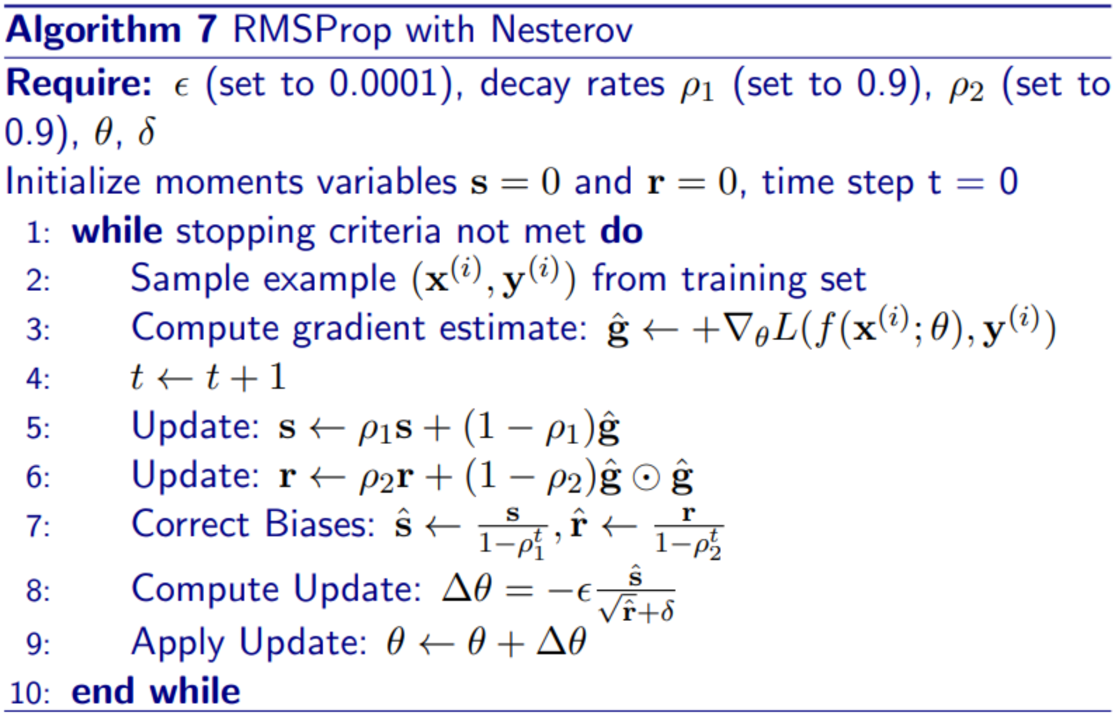
- AdaGrad is good when the objective is convex.
- AdaGrad might shrink the learning rate too aggressively, we want to keep the history in mind
- We can adapt it to perform better in non-convex settings by accumulating an exponentially decaying average of the gradient
- It was proposed in a slide in Geoffrey Hinton's coursera course



```
Algorithm 5 RMSProp
Require: Global Learning rate $\epsilon$, decay parameter $\rho$, $\delta$, Initial Parameter $\theta$
Initialize $\mathbf{r} = 0$
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    Accumulate: $\boxed{\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}}$
    Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```

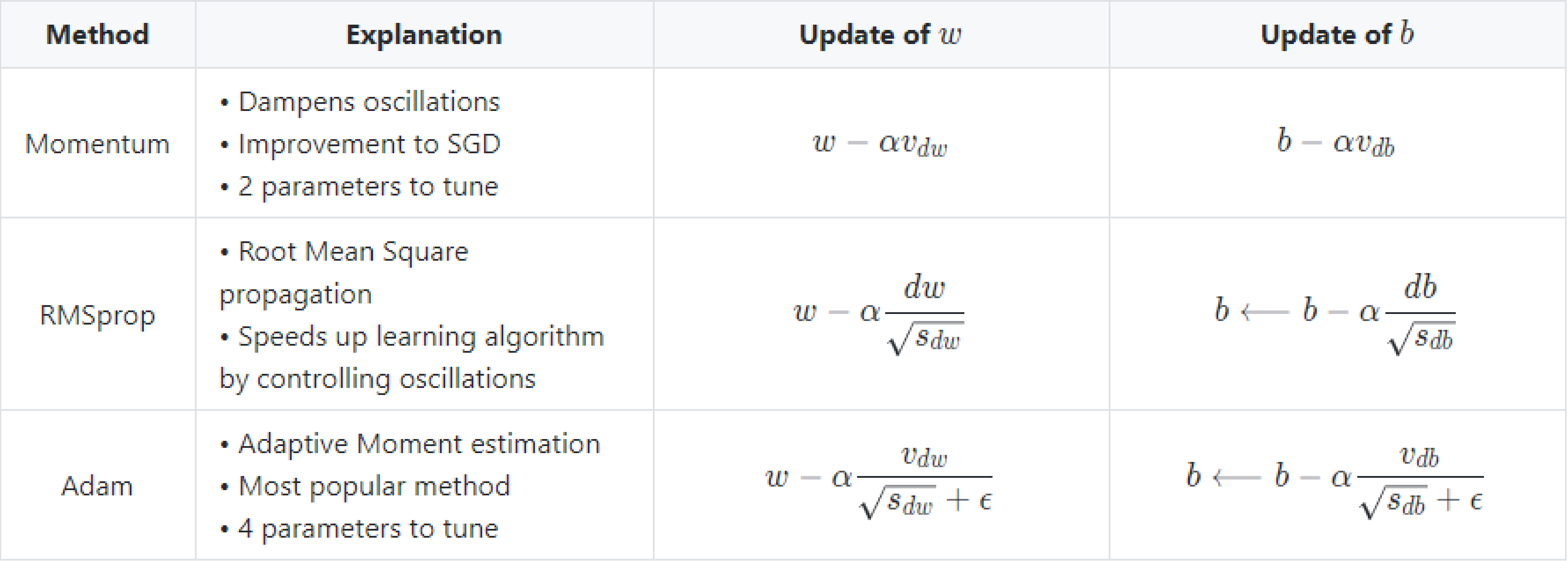
Adam: ADAptive Moments

- We could have used RMSProp with momentum
- Use of Momentum with rescaling is not well motivated
- Adam is like RMSProp with Momentum but with bias correction terms for the first and second moments

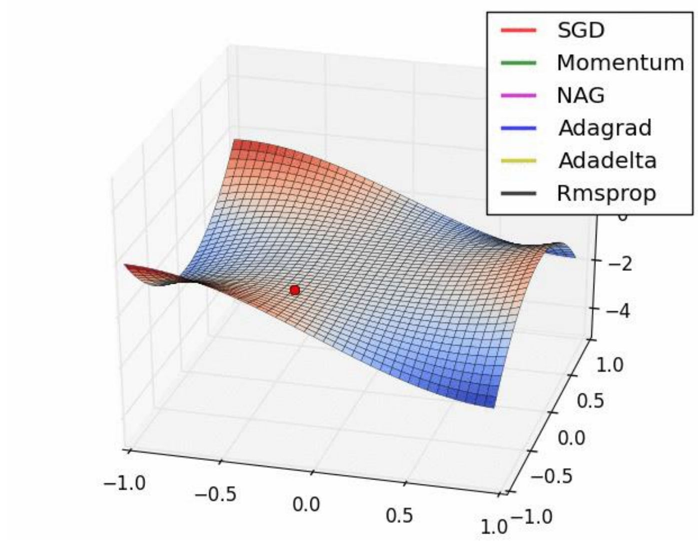


Algorithm 7 RMSProp with Nesterov

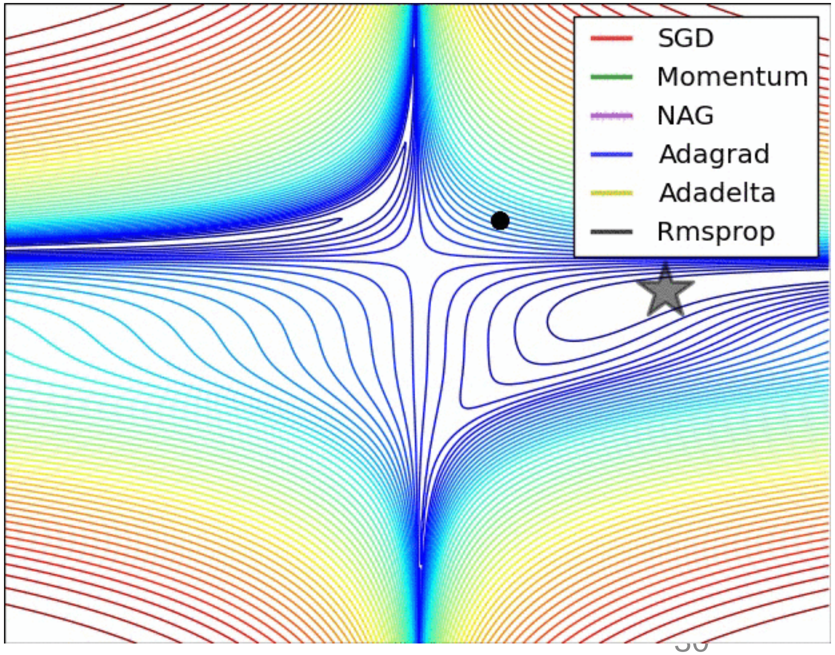
```
Require: $\epsilon$ (set to 0.0001), decay rates $\rho_1$ (set to 0.9), $\rho_2$ (set to 0.9), $\theta$, $\delta$
Initialize moments variables $\mathbf{s} = 0$ and $\mathbf{r} = 0$, time step t = 0
while stopping criteria not met do
    Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    $t \leftarrow t + 1$
    Update: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\hat{\mathbf{g}}$
    Update: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Correct Biases: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\rho_2^t}$
    Compute Update: $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$
    Apply Update: $\theta \leftarrow \theta + \Delta\theta$
end while
```



Method	Explanation	Update of w	Update of b
Momentum	• Dampens oscillations • Improvement to SGD • 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	• Root Mean Square propagation • Speeds up learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	• Adaptive Moment estimation • Most popular method • 4 parameters to tune	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

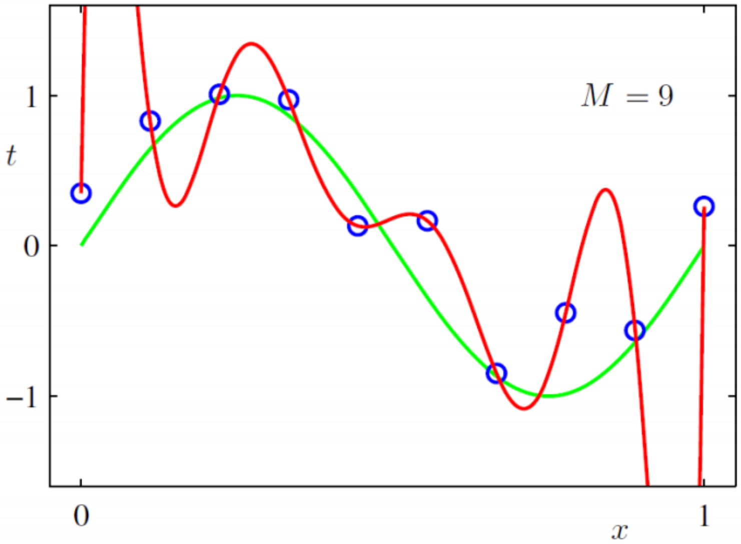


Ref: **Alec Radford**



Regularization

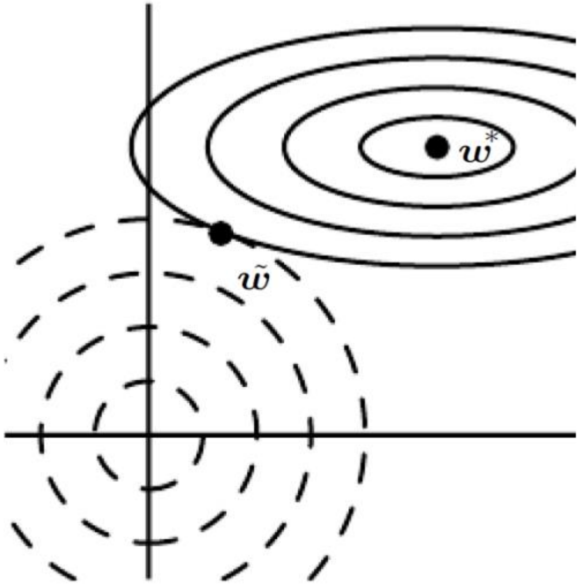
- In general: any method to **prevent overfitting** or help the optimization
- Specifically: additional **terms** in the **training** optimization objective to prevent overfitting or help the optimization



Classical regularization

- L2 regularization
  - **L2 = weight decay**
  - Regularization term that penalizes big weights, added to the objective

$$\min_{\theta} \hat{L}_R(\theta) = \overset{W^*}{\boxed{\hat{L}(\theta)}} + \overset{\tilde{W}}{\boxed{\frac{\alpha}{2}||\theta||_2^2}}$$



- L1 regularization

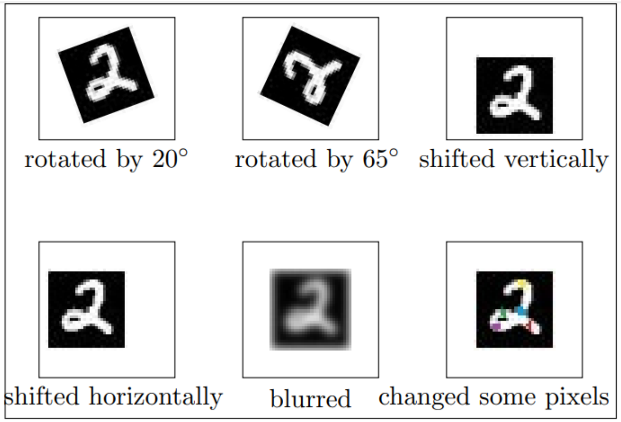
$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \alpha||\theta||_1$$

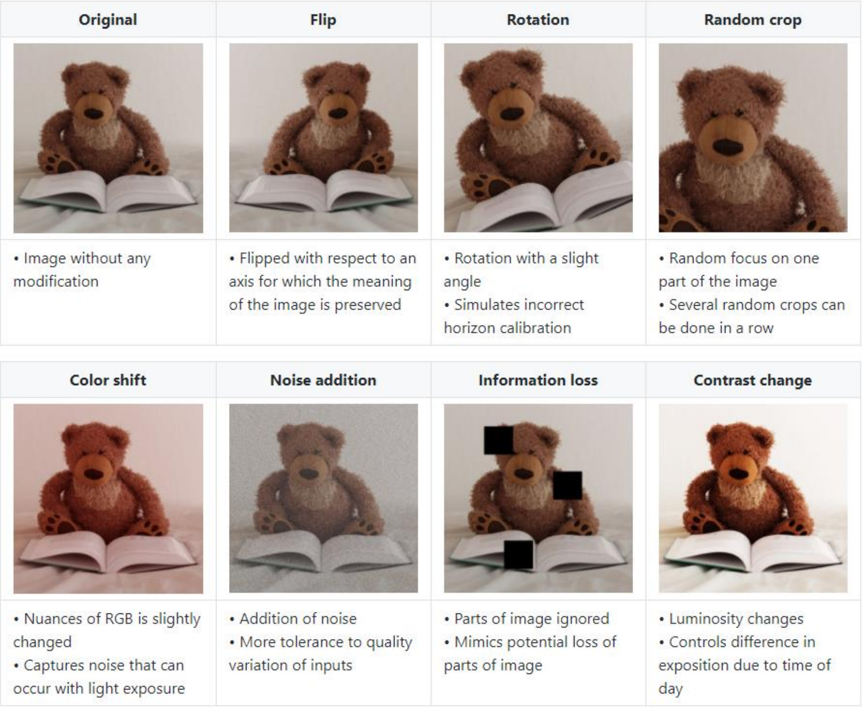
Other types of regularizations

- Data augmentation
- Early stopping
- Dropout
- Batch Normalization

Data augmentation

- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied:
  - Example: classifying ‘b’ and ‘d’ or classifying ‘6’ and ‘9’



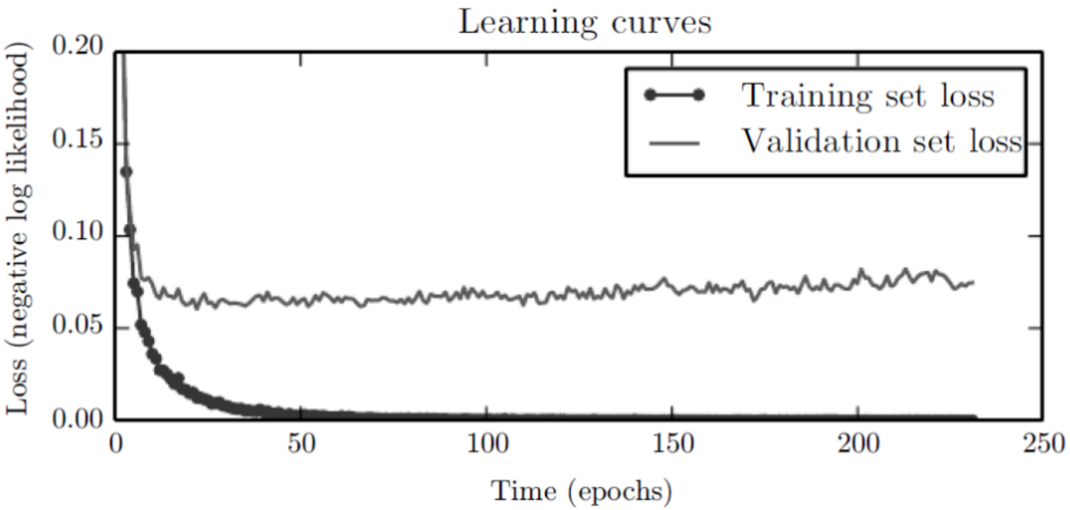


Original	Flip	Rotation	Random crop
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row
Color shift	**Noise addition**	**Information loss**	**Contrast change**
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

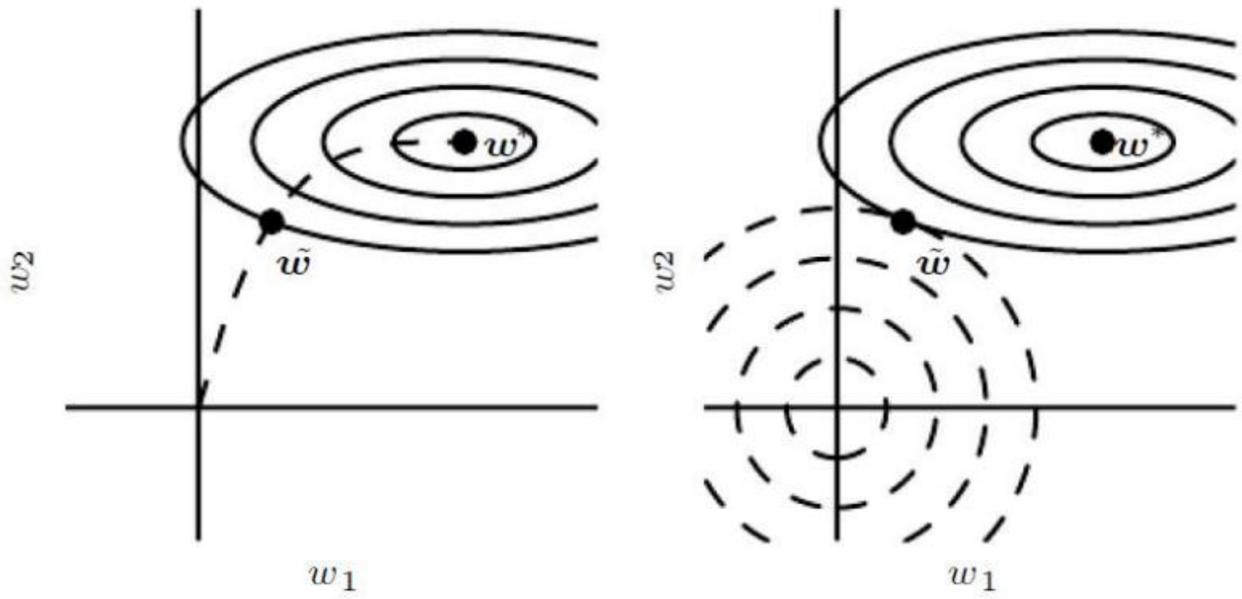
Q: What does data augmentation for structured data?

Early stopping

- Idea: don’t train the network to too small training error
- Recall overfitting: Larger the hypothesis class, easier to find a hypothesis that fits the difference between the two
- Prevent overfitting: do not push the hypothesis too much; use validation error to decide when to stop

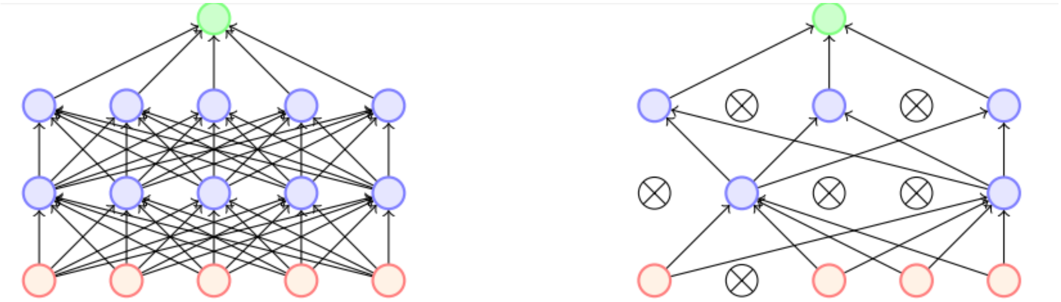


Early stopping as a regularizer



Dropout

- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically p = 0.5) for hidden nodes and p = 0.8 for visible nodes



Batch Normalization (BN)

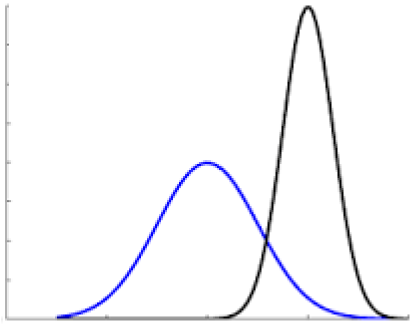
- To normalize the layers' inputs by re-centering and re-scaling.
- It was proposed by Sergey Ioffe and Christian Szegedy in 2015
- *B* to denote a mini-batch of size *m*

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \text{ and } \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2.$$

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}}, \text{ where } k \in [1, d] \text{ and } i \in [1, m]$$

$\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are the per-dimension mean and variance, respectively.

Why BN?

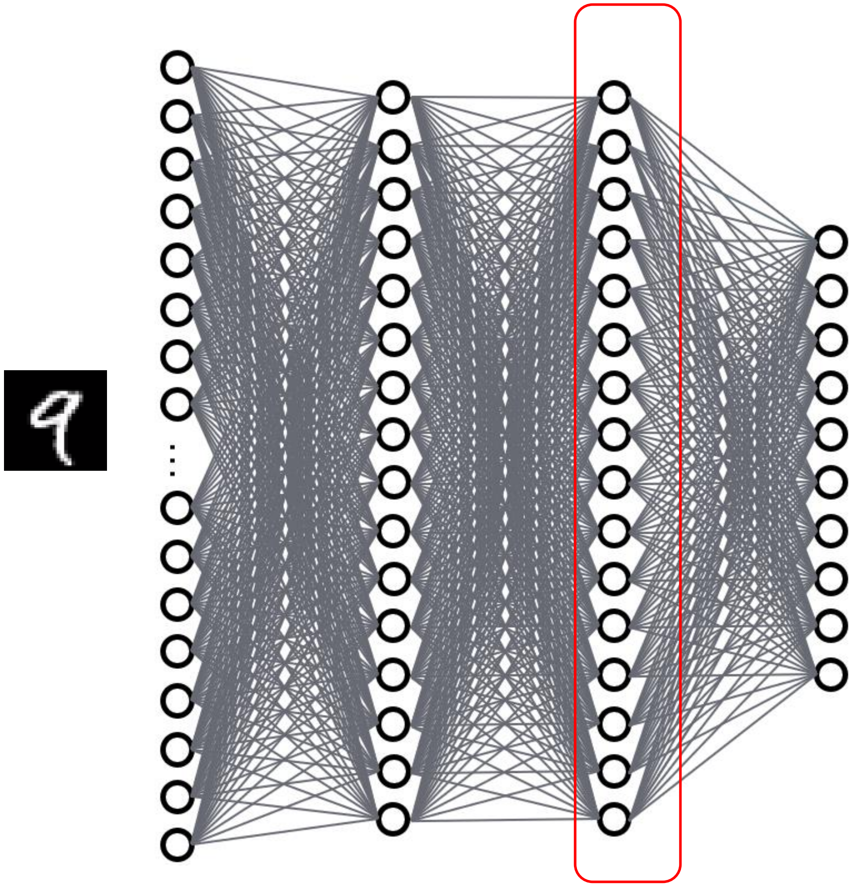


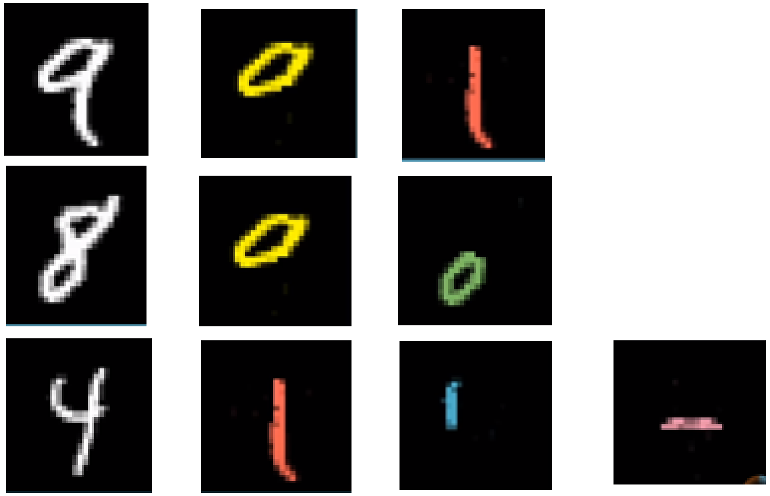
- BN reduces *Covariate Shift*.
  - The change in distribution of activation of a component. By using BN, each neuron's activation becomes (more or less) a Gaussian distribution.
- Covariate Shift is undesirable, because the later layers have to keep adapting to the change of the type of distribution.
- BN reduces effects of exploding and vanishing gradients, because every becomes roughly normal distributed.

BN: Other benefits in practice

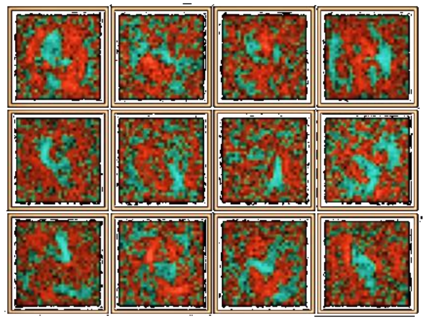
- BN reduces training times.
  - Because of less Covariate Shift, less exploding/vanishing gradients.
- BN reduces demand for regularization, e.g. dropout or L2 norm.
  - Because the means and variances are calculated over batches and therefore every normalized value depends on the current batch. I.e. the network can no longer just memorize values and their correct answers.
- BN allows higher learning rates.
  - Because of less danger of exploding/vanishing gradients.
- BN enables training with saturating nonlinearities in deep networks, e.g. sigmoid.
  - Because the normalization prevents them from getting stuck in saturating ranges, e.g. very high/low values for sigmoid.

Possible Hierarchical Features





But…we got something like this



Let’s get back to this in the next lecture !

Questions?