Deep Learning - Foundations and Concepts

Chapter 7. Gradient Descent

nonlineark@github

February 26, 2025

Outline

Gradient and stationary points

Theorem

Let the function $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable at $a \in \mathbb{R}^n$:

1. Near a the function f increases fastest in the direction of $\nabla f(a) \in \mathbb{R}^n$.
2. The rate of increase in f is measured by the length of $\nabla f(a)$.
3. If f has a local extremum at a then $\nabla f(a) = 0$.

Definition

Let $f : \mathbb{R}^n \to \mathbb{R}$ be differentiable at $a \in \mathbb{R}^n$. Then a is said to be a stationary point for f if $Df(a) = 0$, or, equivalently, $\nabla f(a) = 0$.

Gradient and stationary points

It's easy to see the correctness of the theorem. Since the rate of increase of f at the point a in an arbitray direction $v \in \mathbb{R}^n$ is given by the directional derivative at v, we have:

$$|Df(a)v| = |v^T \nabla f(a)| \leq ||\nabla f(a)||||v||$$

where we have used the Cauchy-Schwarz inequality. The rate of increase is maximal if v is a positive scalar multiple of $\nabla f(a)$. For the third claim, let's define g_j as:

$$g_j(t) = f(a_1, \ldots, a_{j-1}, t, a_{j+1}, \ldots, a_n)$$

then $g_j'(a_j) = D_j f(a)$. Since f has a local extremum at a, g_j also has a local extremum at a_j. Thus $g_j'(a_j) = 0$ for $1 \leq j \leq n$, and we have $\nabla f(a) = 0$.

Gradient and stationary points

During training, we want to optimize the weights and biases $w \in \mathbb{R}^W$ by using a chosen error function $E(w)$. From the previous theorem we see that, its smallest value will occur at a point in weight space such that:

$$\nabla E(w) = 0$$

But:

- Global minimum vs. local minimum.
- For any point w that is a local minimum, there will generally be other points in weight space that are equivalent minima (weight-space symmetries).

Local quadratic approximation

Threorem

Let U be a convex open subset of $\mathbb{R}^n$ and let $a \in U$ be a stationary point for $f \in C^2(U)$. Then we have the following assertions:

1. If $Hf(a)$ is positive definite, then f has a local strict minimum at a.
2. If $Hf(a)$ is negative definite, then f has a local strict maximum at a.

where $Hf(a)$ is the Hessian of f at a.

Local quadratic approximation

We only prove the first claim. Using Taylor expansion, we see that:

$$f(a+h) = f(a) + h^T \nabla f(a) + \frac{1}{2} h^T H f(a) h + R_2(a,h)$$
$$= f(a) + \frac{1}{2} h^T H f(a) h + R_2(a,h)$$

where $\lim_{h \to 0} \frac{R_2(a,h)}{||h||^2} = 0$. Since $f \in C^2(U)$, $Hf(a)$ is a self-adjoint operator. Let λ be its smallest eigenvalue. Because $Hf(a)$ is positive definite, $\lambda > 0$. Notice that:

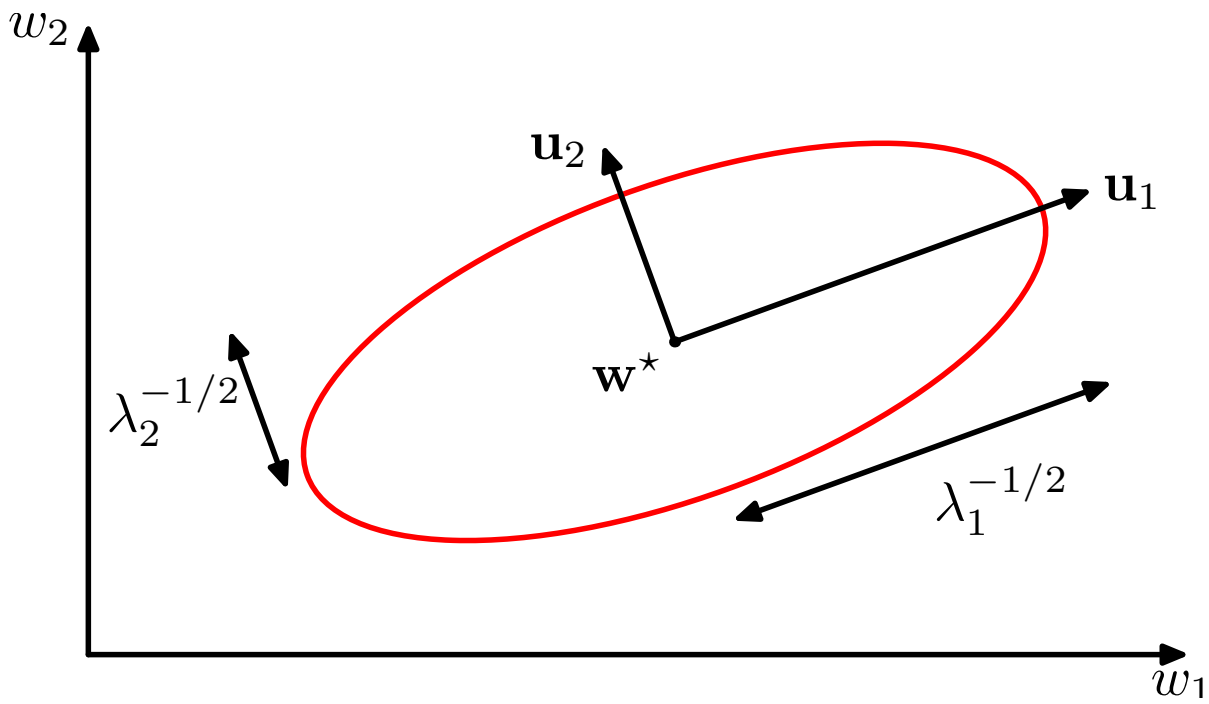
- $h^T H f(a) h \geq \lambda ||h||^2$.
- There is $\delta > 0$, such that $\frac{|R_2(a,h)|}{||h||^2} < \frac{\lambda}{4}$ for $||h|| < \delta$.

For $||h|| < \delta$, we have:

$$f(a+h) - f(a) = \frac{1}{2} h^T H f(a) h + R_2(a,h) > \frac{\lambda}{2} ||h||^2 - \frac{\lambda}{4} ||h||^2 = \frac{\lambda}{4} ||h||^2$$

From the previous theorem, we see that: A necessary and sufficient condition for w^* to be a local minimum of the error function $E(w)$ is that the gradient of $E(w)$ should vanish at w^* and the Hessian matrix evaluated at w^* should be positive definite.

Figure: Geometry of the error surface in the neighbourhood of a minimum w^*



Use of gradient information

There is no way we can find an analytical solution to the equation $\nabla E(w) = 0$, so we resort to iterative numerical procedures to minimize the error function:

$$w^{(\tau)} = w^{(\tau-1)} + \Delta w^{(\tau-1)}$$

Use of gradient information

Let's estimate the computational effort needed. The Taylor expansion of $E(w)$ around some point $\hat{w}$ in weight space is given by:

$$E(w) \approx E(\hat{w}) + (w - \hat{w})^T \nabla E(\hat{w}) + \frac{1}{2}(w - \hat{w})^T HE(\hat{w})(w - \hat{w})$$

We see that in order to locate the minimum, we need at least $\mathcal{O}(W^2)$ pieces of information, because:

- $\nabla E(\hat{w})$ contains W independent parameters.
- $HE(\hat{w})$ contains $\frac{W(W+1)}{2}$ independent parameters.

Use of gradient information

- If we do not make use of gradient information, we would expect to have to perform $\mathcal{O}(W^2)$ function evaluations, each of which would require $\mathcal{O}(W)$ steps. Thus the total cost would be $\mathcal{O}(W^3)$.
- If we make use of the gradient information, each evaluation of ∇E brings W pieces of information, so we would expect to find the minimum of the function in $\mathcal{O}(W)$ gradient evaluations. In the next chapter we shall see, by using error backpropagation, each such evaluation takes only $\mathcal{O}(W)$ steps. Thus the total cost would be $\mathcal{O}(W^2)$.

Batch gradient descent

The simplest approach to using gradient information is to take a small step in the direction of the negative gradient:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \nabla E(w^{(\tau-1)})$$

where the parameter $\eta > 0$ is known as the learning rate. At each step:

- The weight vector is moved in the direction of the greatest rate of decrease of the error function ("gradient descent" in the name).
- The entire training set will be processed ("batch" in the name).

Stochastic gradient descent

Note that error functions based on maximum likelihood for a set of independent observations comprise a sum of terms: $E(w) = \sum_{n=1}^{N} E_n(w)$. We can update the weight vector based on one data point at a time:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \nabla E_n(w^{(\tau-1)})$$

Advantages of stochastic gradient descent:

- It is much more efficient on very large data sets.
- It works when the data arises from a continuous stream of new data points (a.k.a., online algorithm).
- It handles redundancy in the data much more efficiently.
- The possibility of escaping from local minima.

Stochastic gradient descent

Algorithm 1: Stochastic gradient descent

$n \leftarrow 1$;

repeat

  $w \leftarrow w - \eta \nabla E_n(w)$;

  $n \leftarrow n + 1 (\mathrm{mod} N)$;

until *convergence*;

return w;

Mini-batches

Algorithm 2: Mini-batch stochastic gradient descent

$n \leftarrow 1$;
repeat
  $w \leftarrow w - \eta \nabla E_{n:n+B-1}(w)$;
  $n \leftarrow n + B$;
  if $n > N$ **then**
    shuffle data;
    $n \leftarrow 1$;
  end
until *convergence*;
return w;

Parameter initialization

The initial setting for the parameters being learned can have a significant effect on:

- How long it takes to reach a solution.
- The generalization performance of the resulting trained network.

Although there is relatively little theory to guide the initialization strategy, one key consideration is symmetry breaking, by initializing parameters randomly from some distribution:

- Uniform distribution in the range $[-\epsilon, \epsilon]$.
- Zero-mean Gaussian of the form $\mathcal{N}(0, \epsilon^2)$.

Parameter initialization

The approach of He initialization for choosing the value of ϵ. Consider a network in which layer l evaluates the following transformations:

$$a_i^{(l)} = \sum_{m=1}^{M} w_{im} z_m^{(l-1)}$$

$$z_i^{(l)} = \mathrm{ReLU}(a_i^{(l)})$$

Suppose we initialize the weights using a Gaussian $\mathcal{N}(0, \epsilon^2)$, and suppose that the pre-activations $a_m^{(l-1)}$ in layer $l-1$ have variance λ^2.

Parameter initialization

We see that:

$$E(a_i^{(l)}) = \sum_{m=1}^{M} E(w_{im})E(z_m^{(l-1)}) = 0$$

$$\begin{aligned}
\mathrm{var}(a_i^{(l)}) &= E((a_i^{(l)})^2) - (E(a_i^{(l)}))^2 = E((a_i^{(l)})^2) \\
&= E(\sum_{1 \le m,n \le M} w_{im} w_{in} z_m^{(l-1)} z_n^{(l-1)}) = E(\sum_{m=1}^{M} w_{im}^2 (z_m^{(l-1)})^2) \\
&= \sum_{m=1}^{M} E(w_{im}^2) E((z_m^{(l-1)})^2) = \sum_{m=1}^{M} \epsilon^2 \frac{1}{2} E((a_m^{(l-1)})^2) \\
&= \sum_{m=1}^{M} \epsilon^2 \frac{1}{2} \lambda^2 = \frac{M}{2} \epsilon^2 \lambda^2
\end{aligned}$$

Parameter initialization

If we require that the pre-activations at layer l also have variance λ^2 then we arrive at the following choice for ϵ to initialize the weights that feed into a unit with M inputs:

$$\epsilon = \sqrt{\frac{2}{M}}$$

The bias parameters are typically set to small positive values to ensure that most pre-activations are initally active during learning.

The rate of convergence

When applying gradient descent in practice, we need to choose a value for the learning parameter η:

- Increasing the value of η may lead to oscillations, and if we increase η too much, those oscillations will become divergent.
- So η must be kept sufficiently small, gradient descent then takes many small steps to reach the minimum and is a very inefficient procedure.

The rate of convergence

Consider the local quadratic approximation around a point w^* that is a minimum of the error function:

$$E(w) = E(w^*) + \frac{1}{2}(w - w^*)^T HE(w^*)(w - w^*)$$
$$\nabla E(w) = HE(w^*)(w - w^*)$$

And there is an orthogonal matrix U that will diagonalize $HE(w^*)$: $\Lambda = U^T HE(w^*)U$. Under this new orthonormal basis where the transformation is given by $w - w^* = Uu$, we see that:

$$\Delta u = U^T \Delta w = U^T(-\eta \nabla E(w)) = -\eta U^T HE(w^*)(w - w^*)$$
$$= -\eta U^T (U \Lambda U^T) U u = -\eta \Lambda u$$

The rate of convergence

For the ith coordinate, this is simply:

$$\Delta u_i = -\eta \lambda_i u_i$$
$$u_i^{\text{new}} = (1 - \eta\lambda_i) u_i^{\text{old}}$$
$$u_i^{(T)} = (1 - \eta\lambda_i)^T u_i^{(0)}$$

It follows that, provided $|1 - \eta\lambda_i| < 1$, the limit $T \to \infty$ leads to $u_i = 0$, which means $w = w^*$, and so the weight vector has reached the minimum of the error. Further we see that:

- Convergence to the stationary point requires that all the λ_i be positive.
- The value of η has to be $\eta < \frac{2}{\lambda_{\max}}$ where $\lambda_{\max}$ is the largest of the eigenvalues.
- The rate of convergence is dominated by the smallest eigenvalue.

Momentum

One simple technique for dealing with the problem of widely differing eigenvalues is to add a momentum term to the gradient descent formula:

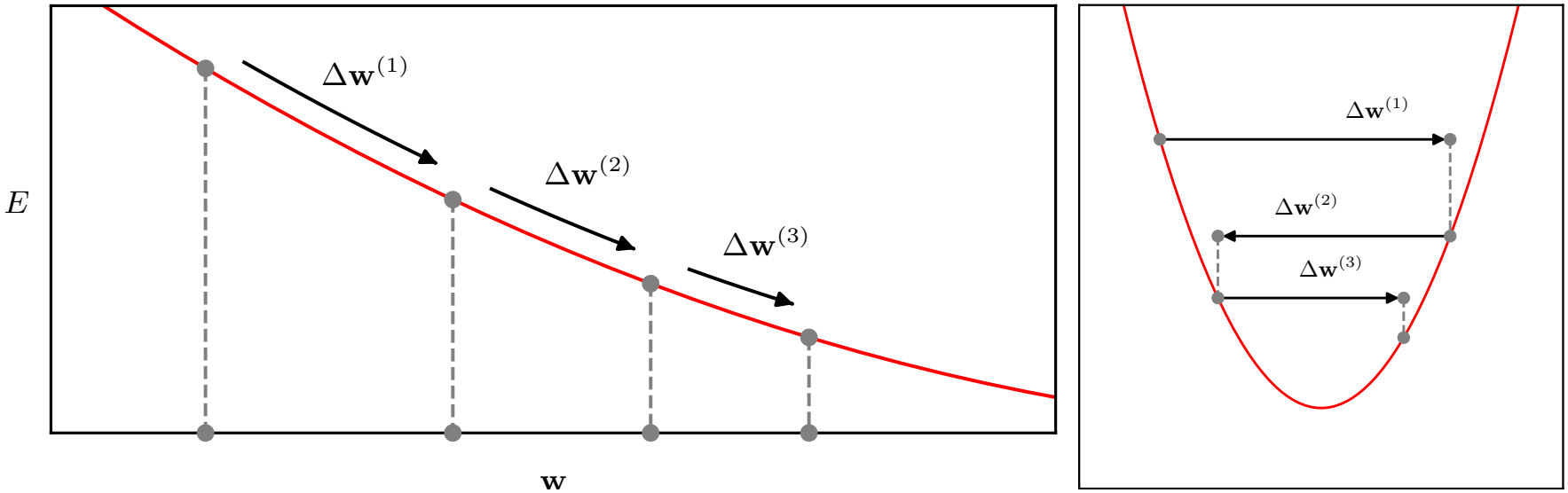
$$\Delta w^{(\tau-1)} = -\eta \nabla E(w^{(\tau-1)}) + \mu \Delta w^{(\tau-2)}$$

To understand the effect of the momentum term, let's consider two cases:

- If the error surface has relatively low curvature ($\nabla E(w)$ is almost constant): $\Delta w = -\eta \nabla E(1 + \mu + \mu^2 + \cdots) = -\frac{\eta}{1-\mu} \nabla E$. We see the effective learning rate is increased from η to $\frac{\eta}{1-\mu}$.
- If the error surface has high curvature in which the gradient descent is oscillatory, successive contributions from the momentum term will tend to cancel and the effective learning rate will be close to η.

Momentum

Figure: The effect of the momentum term



Momentum

Algorithm 3: Stochastic gradient descent with momentum

$n \leftarrow 1$;
$\Delta w \leftarrow 0$;
repeat
  $\Delta w \leftarrow -\eta \nabla E_{n:n+B-1}(w) + \mu \Delta w$;
  $w \leftarrow w + \Delta w$;
  $n \leftarrow n + B$;
  if $n > N$ **then**
    suffle data;
    $n \leftarrow 1$;
  end
until *convergence*;
return w;

Learning rate schedule

To specify a value for the learning rate parameter η, in practice, the best results are obtained by using a larger value for η at the start of training and then reducing the learning rate over time:

$$w^{(\tau)} = w^{(\tau-1)} - \eta^{(\tau-1)} \nabla E_n(w^{(\tau-1)})$$

Here are some examples of learning rate schedules:

$$\eta^{(\tau)} = (1 - \frac{\tau}{K})\eta^{(0)} + \frac{\tau}{K}\eta^{(K)}$$

$$\eta^{(\tau)} = \eta^{(0)}(1 + \frac{\tau}{s})^c$$

$$\eta^{(\tau)} = \eta^{(0)} c^{\frac{\tau}{s}}$$

RMSProp and Adam

The optimal learning rate depends on the local curvature of the error surface. There are several algorithms that use different learning rates for each parameter in the network.

AdaGrad (adaptive gradient):

$$r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial E(w)}{\partial w_i^{(\tau-1)}}\right)^2$$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)}} + \delta} \frac{\partial E(w)}{\partial w_i^{(\tau-1)}}$$

RMSProp and Adam

RMSProp (root mean square propagation):

$$r_i^{(\tau)} = \beta r_i^{(\tau-1)} + (1-\beta)\left(\frac{\partial E(w)}{\partial w_i^{(\tau-1)}}\right)^2$$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)}} + \delta} \frac{\partial E(w)}{\partial w_i^{(\tau-1)}}$$

where $0 < \beta < 1$ and a typical value is $\beta = 0.9$.

RMSProp and Adam

Adam (adaptive moments):

$$s_i^{(\tau)} = \beta_1 s_i^{(\tau-1)} + (1-\beta_1)\frac{\partial E(w)}{\partial w_i^{(\tau-1)}}$$

$$r_i^{(\tau)} = \beta_2 r_i^{(\tau-1)} + (1-\beta_2)(\frac{\partial E(w)}{\partial w_i^{(\tau-1)}})^2$$

$$\hat{s}_i^{(\tau)} = \frac{s_i^{(\tau)}}{1-\beta_1^{\tau}}$$

$$\hat{r}_i^{(\tau)} = \frac{r_i^{(\tau)}}{1-\beta_2^{\tau}}$$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{r}_i^{(\tau)}} + \delta}\hat{s}_i^{(\tau)}$$

where $0 < \beta_1, \beta_2 < 1$ and typical values for them are $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

RMSProp and Adam

Algorithm 4: Adam optimization

$n \leftarrow 1, \tau \leftarrow 1, s \leftarrow 0, r \leftarrow 0;$

repeat

  $g = \nabla E_{n:n+B-1}(w);$

  $s \leftarrow \beta_1 s + (1-\beta_1) g, r \leftarrow \beta_2 r + (1-\beta_2) g * g;$

  $\hat{s} \leftarrow \frac{s}{1-\beta_1^\tau}, \hat{r} \leftarrow \frac{r}{1-\beta_2^\tau};$

  $w \leftarrow w - \frac{\eta}{\sqrt{\hat{r}}+\delta} \hat{s};$

  $n \leftarrow n + B, \tau \leftarrow \tau + 1;$

  if $n > N$ **then**

    shuffle data;

    $n \leftarrow 1;$

  end

until *convergence*;

return w;

Data normalization

For continuous input variables, it can be very beneficial to re-scale the input values so that they span similar ranges:

$$\mu_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n$$

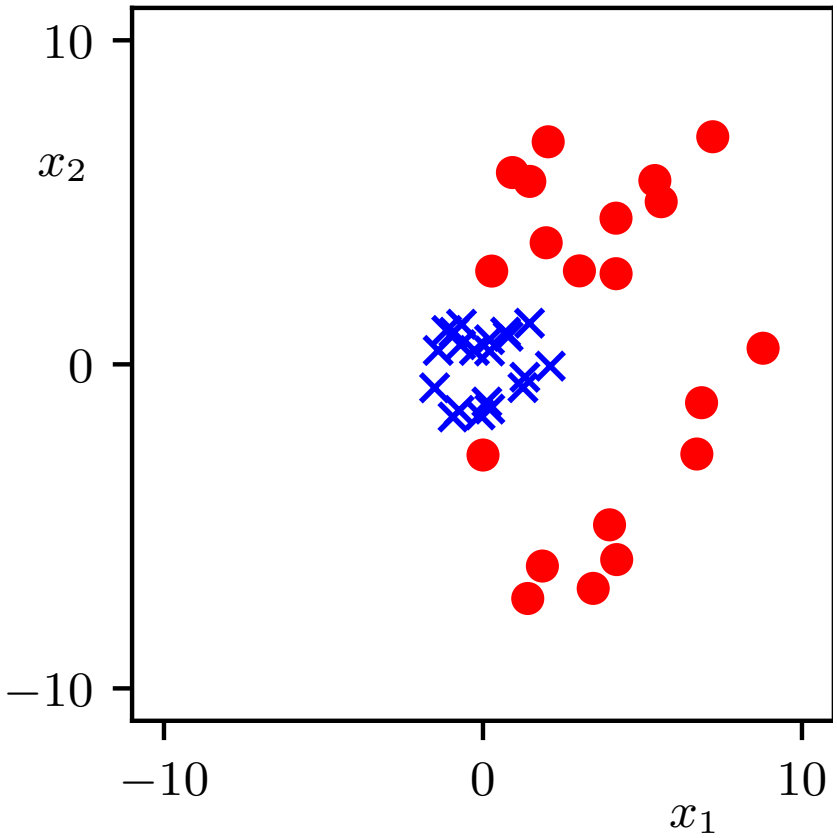
$$\sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N}(x_i^n - \mu_i)^2$$

$$\tilde{x}_i^n = \frac{x_i^n - \mu_i}{\sigma_i}$$

This way the re-scaled values $\{\tilde{x}_i^n\}$ have zero mean and unit variance.

Data normalization

Figure: Illustration of input data normalization



Batch normalization

- If there is wide variation in the range of activation values in a particular hidden layer, then normalizing those values to have zero mean and unit variance should make the learning problem easier for the next layer.
- Batch normalization largely resolves the issues of vanishing gradients and exploding gradients.
- We can either normalize the pre-activation values a_i or the activation values z_i.

Batch normalization

For a mini-batch of size K:

$$\mu_m = \frac{1}{K}\sum_{k=1}^{K} a_m^k$$

$$\sigma_m^2 = \frac{1}{K}\sum_{k=1}^{K}(a_m^k - \mu_m)^2$$

$$\hat{a}_m^k = \frac{a_m^k - \mu_m}{\sqrt{\sigma_m^2 + \delta}}$$

$$\tilde{a}_m^k = \gamma_m \hat{a}_m^k + \beta_m$$

where β_m and γ_m are adaptive parameters that are learned by gadient descent to compensate the reduction of the layer's representational capability caused by the normalization.

Batch normalization

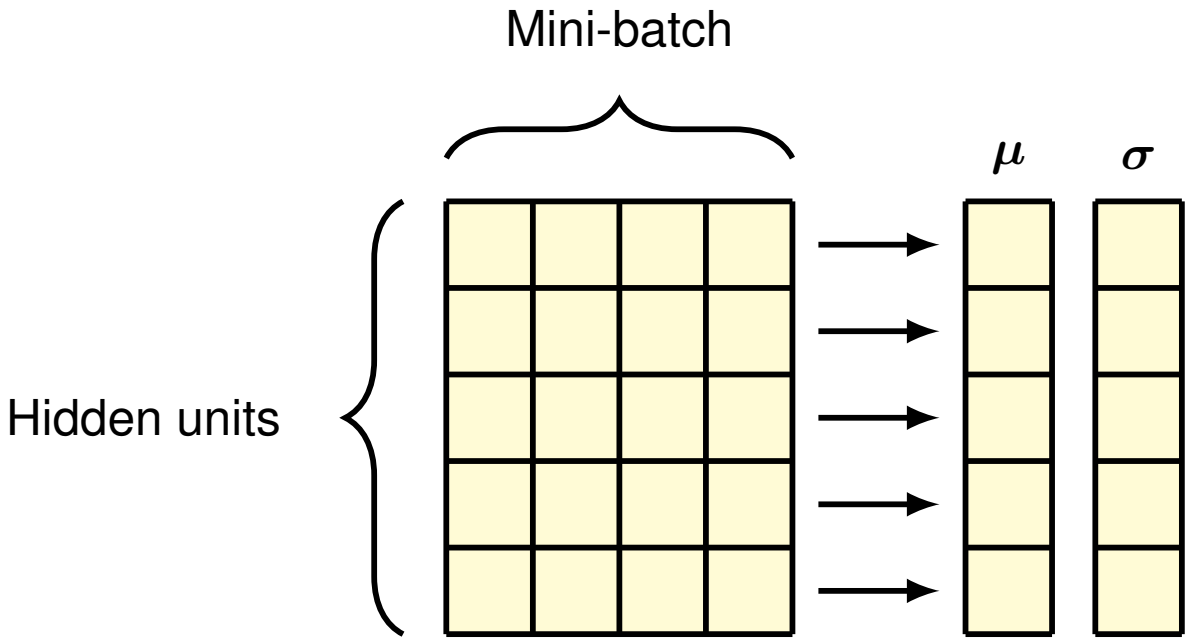
During the inference phase, the training mini-batches are no longer available, and we cannot determine a mean and variance from just one data example. To solve this, we compute moving averages throughout the training phase:

$$\bar{\mu}_i^{(\tau)} = \alpha \bar{\mu}_i^{(\tau-1)} + (1-\alpha)\mu_i$$
$$\bar{\sigma}_i^{(\tau)} = \alpha \bar{\sigma}_i^{(\tau-1)} + (1-\alpha)\sigma_i$$

where $0 \leq \alpha \leq 1$. These moving averages are used to process new data points during the inference phase.

Batch normalization

Figure: Illustration of batch normalization



Layer normalization

An alternative to normalizing across examples within a mini-batch for each hidden unit separately is to normalize across the hidden-unit values for each data point separately. For a layer with M hidden units:

$$\mu_k = \frac{1}{M} \sum_{m=1}^{M} a_m^k$$

$$\sigma_k^2 = \frac{1}{M} \sum_{m=1}^{M} (a_m^k - \mu_k)^2$$

$$\hat{a}_m^k = \frac{a_m^k - \mu_k}{\sqrt{\sigma_k^2 + \delta}}$$

$$\tilde{a}_m^k = \gamma_m \hat{a}_m^k + \beta_m$$

Note that the same normalization function can be employed during training and during inference.

Layer normalization

Figure: Illustration of layer normalization