Deep Learning

Lecture 2

Recap

- Multi-layer perceptron
  - Motivation
  - Activation function
  - Properties
- Gradient calculation
  - Numerical calculation
  - Automatic differentiation
  - Manual differentiation

Content

How to use a gradient for optimization?

- Classical gradient descent
- Modified gradient descent
- Regularization techniques

Neural network optimization

Optimization problem

Solving any machine learning requires function optimization. Let F some loss function: L2 loss, cross-entropy, etc.

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \to \min_x, n \gg 1, x \in \mathbb{R}^q$$

We need some optimization method. What about gradient descent?

Function	Calculation cost
$f_i(x)$	$\mathcal{O}(q)$
$\nabla f_i(x)$	$\mathcal{O}(q)$
$F(x)$	$\mathcal{O}(nq)$
$\nabla F(x)$	$\mathcal{O}(nq)$

When n is big, we have problems…

Stochastic gradient descent

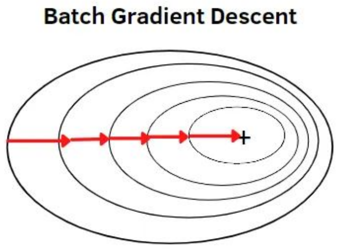
Solution: what if you use not all samples, but just one?

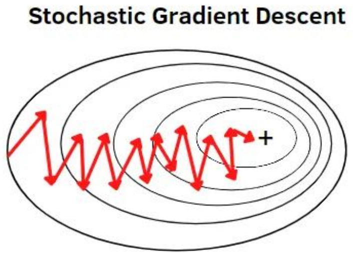
$$
\begin{aligned}
i_k &\sim Unif(1, 2, \ldots, n) \\
g_k &= \nabla f_{ik}(x_k) \\
x_{k+1} &= x_k - \alpha_k g_k
\end{aligned}
$$

Expected value of gradient is equal to the true gradient!

$$\mathbb{E} g_k = \nabla F(x_k)$$

The problem is - high variance of gradient





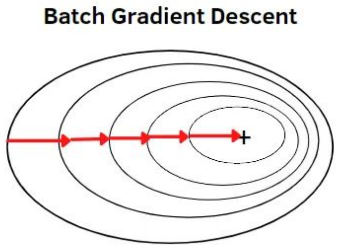
Batch SGD

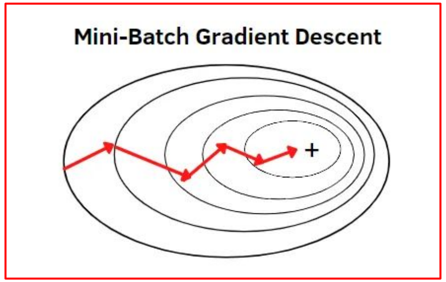
Instead of one sample we can take several. Then, we variance of the gradient is reduced.

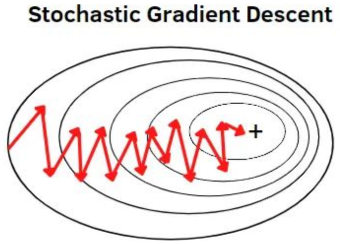
$$I_k \subset Unif(1, 2, \ldots, n)$$

$$g_k = \frac{1}{|I_k|} \sum_{i \in I_k} \nabla f_{ik}(x_k)$$

$$x_{k+1} = x_k - \alpha_k g_k$$







Expected value of gradient is equal to the true gradient!

$$\mathbb{E} g_k = \nabla F(x_k)$$

What happens with optimization close to minima?

To understand what really happen close to minima, let’s consider the following example:

$$F(w) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - wx_i)^2$$

Consider three function:

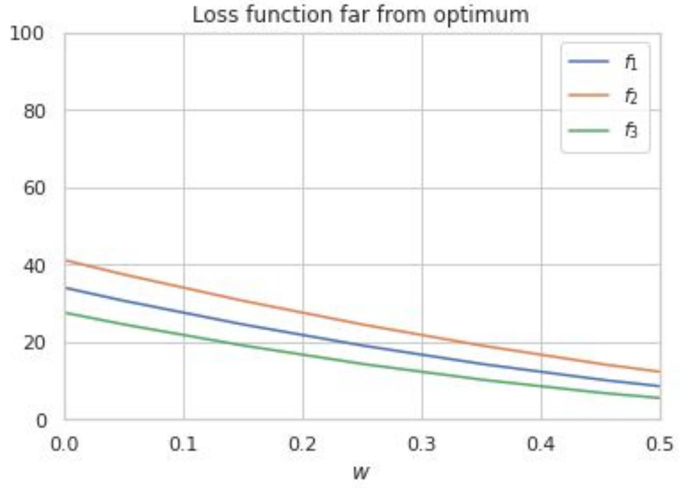
$$y_1(x) = x \qquad y_2(x) = 0.9x$$

$$y_3(x) = 1.1x$$

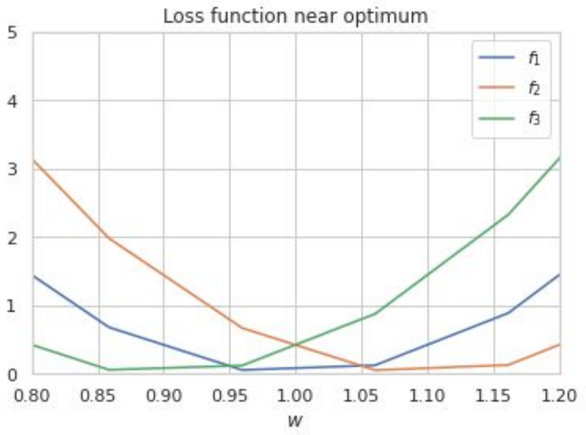
Let’s figure out what happens if we run SGD near optima

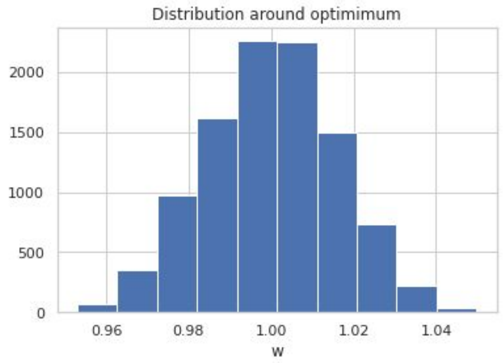
Convergence

Far from optima, the gradient is the same



Close to optima, the gradient of different example starts to contradict





Theorem

The convergence of convex function under some constraints can be explained by the following formula:

$$\mathbb{E}F(\bar{x}_k) - F_{opt} \leqslant \frac{\|x_o - x_{opt}\|^2 + \sum_{i=1}^{k} \alpha_i^2 \mathbb{E}\|g_i\|^2}{2(\sum_{i=1}^{k} \alpha_i)} \qquad \bar{x}_k = \frac{\sum_i \alpha_i x_i}{\sum_i \alpha_i}$$

Proof

Tradeoff of learning rate

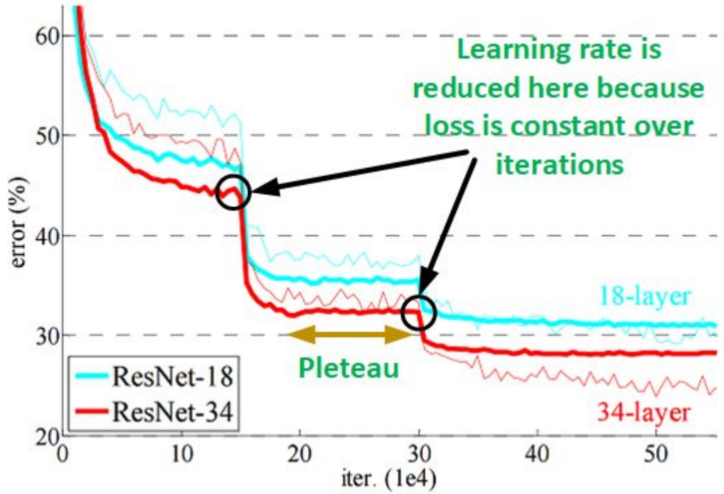
$$\mathbb{E}F(\bar{x}_k) - F_{opt} \leqslant \frac{\|x_o - x_{opt}\|^2 + \sum_{i=1}^{k} \alpha_i^2 \mathbb{E}\|g_i\|^2}{2(\sum_{i=1}^{k} \alpha_i)}$$

Let $R = \|x_0 - x_{opt}\|^2$

$$G = \mathbb{E}\|g_i\|^2$$

$$\alpha_i = h$$

Then: $$\frac{R^2 + G^2(k+1)h^2}{2(k+1)h} = \frac{R^2}{2(k+1)h} + \frac{G^2 h}{2}$$



Learning rate constraints

$$\mathbb{E}F(\bar{x}_k) - F_{opt} \leqslant \frac{\|x_o - x_{opt}\|^2 + \sum_{i=1}^{k} \alpha_i^2 \mathbb{E}\|g_i\|^2}{2(\sum_{i=1}^{k} \alpha_i)}$$

Learning rate constraints:

$$\sum_{i=1} \alpha_i = \infty$$

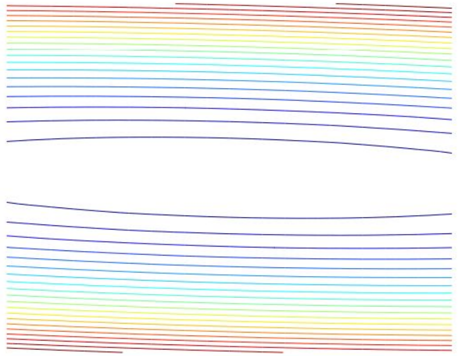
$$\sum_{i=1} \alpha_i^2 < \infty$$

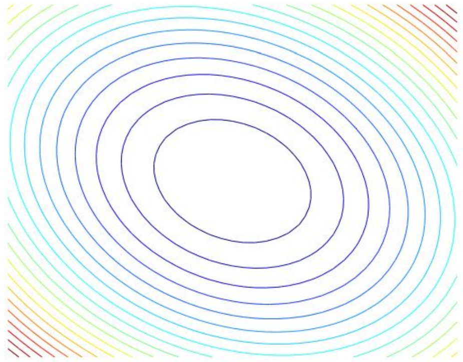
Adaptive methods motivation

To understand how we can modify gradient descent to achieve faster convergence. Let's consider the simple model

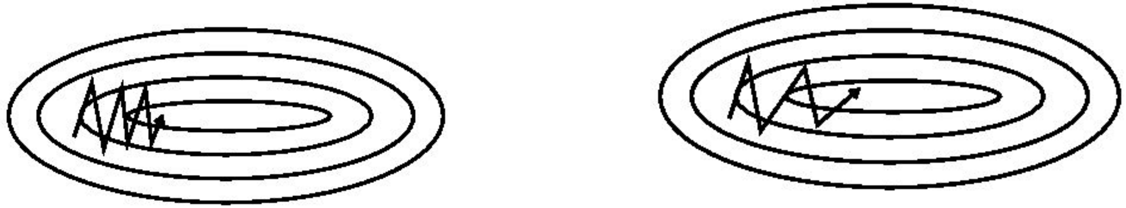
$$f(x) = \frac{1}{2}x^\top Ax - x^\top b \to \min_x; A = A^\top$$

This model is quite reasonable because near optima we can use Taylor series till second order and we will get this quadratic form





Momentum

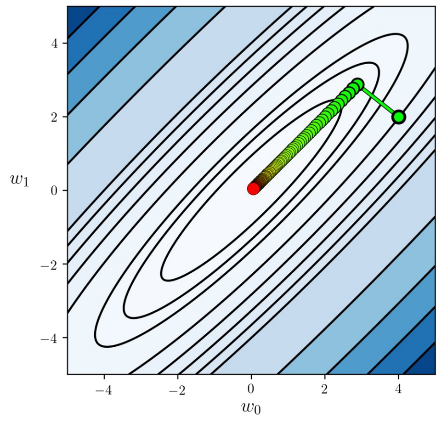


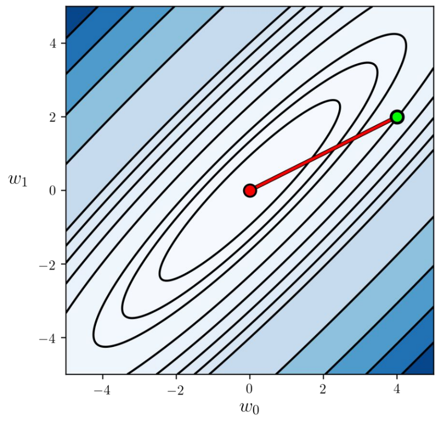
$$x_{k+1} = x_k - \alpha_k g_k + \beta_k (x_k - x_{k-1})$$

Why does momentum work?

Perfect solution

What is the best method for quadratic forms?





Newton method!

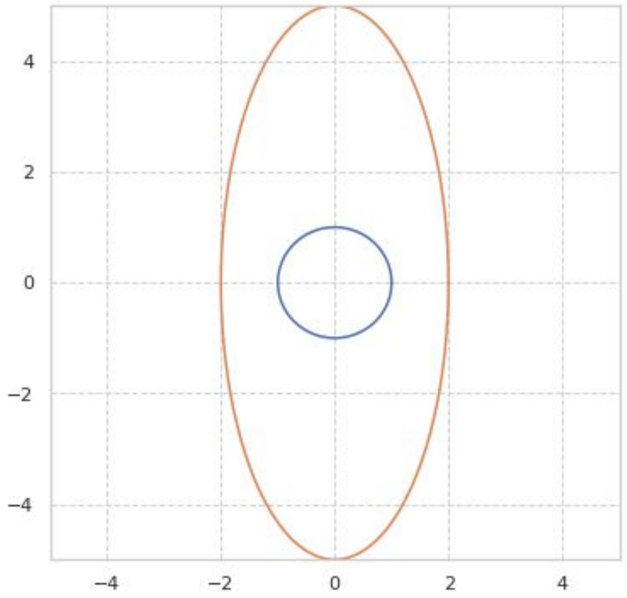
$$x_{k+1} = x_k - \alpha_k (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

AdaGrad

Can we make up method which will be close to Newton method? To get even convergence over several axis. Let’s normalize gradient w.r.t. to its norm over this direction.

$$x_{k+1,i} = x_{k,i} - \alpha_k \frac{g_{k,i}}{\sqrt{v_{k,i} + \varepsilon}}$$

$$v_{k,i} = \sum_{j=0}^{k} g_{k,i}^2$$



After some steps we will have equal convergence over all axis. As a result, our quadratic form will start looks like as on picture

RMSProp

But if we continue optimization for too long the normalization will become too big and convergence will slow down

$$v_{k,i} = \sum_{j=0}^{k} g_{k,i}^2$$

As alternative, we can use momentum to prevent convergence from slowing down

$$x_{k+1,i} = x_{k,i} - \alpha_k \frac{g_{k,i}}{\sqrt{v_{k,i} + \varepsilon}}$$

$$v_{k,i} = \beta v_{k-1,i} + (1 - \beta) g_{k,i}^2$$

Adam

How to make the perfect algorithm? Combine the best ideas: Adam = RMSProp + Momentum

$$x_{k+1,i} = x_{k,i} - \alpha_k \frac{\mu_{k,i}}{\sqrt{v_{k,i} + \varepsilon}}$$

$$v_{k,i} = \beta v_{k-1,i} + (1-\beta) g_{k,i}^2$$

$$\mu_{k,i} = \beta_2 \mu_{k-1,i} + (1-\beta_2) g_{k,i}$$

Add link to some paper or visualization

Сравнение SGD, Momentum, RMSProp, Adam

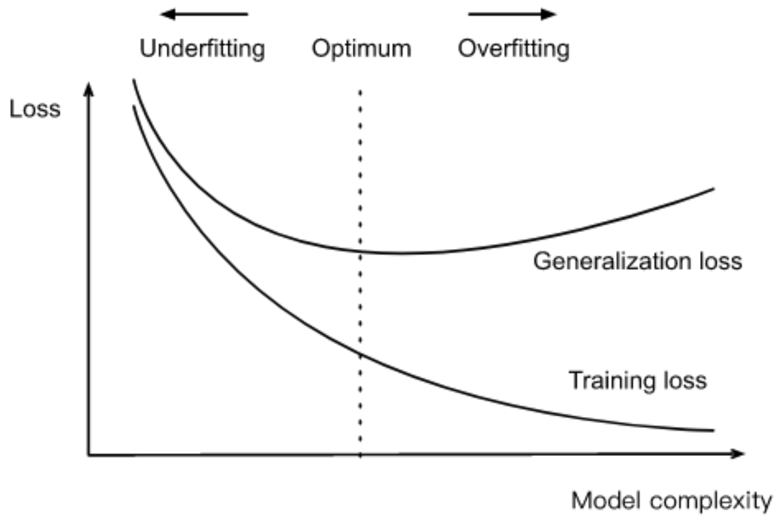
Визуализация

Современные оптимизатор: LocoProp

Основная идея - отдельная оптимизация весов каждого слоя сети

Статья, блог-пост в DLStories

Model regularization



From machine learning, we know that model complexity affects generalization ability of the model. If the model is too complex, (neural networks). So, it has to be regularized to achieve better generalization

About double descent https://mlu-explain.github.io/double-descent
Grokking: link

Weight decay

The first technique for model regularization which comes to mind is L2 Loss regularization. In deep learning, it's called **weight decay.**

$$L(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2,$$

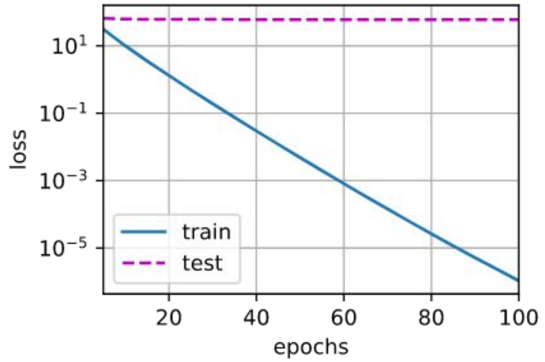
If we write out the equation for weight updates, we can directly understand why it's weight decay.

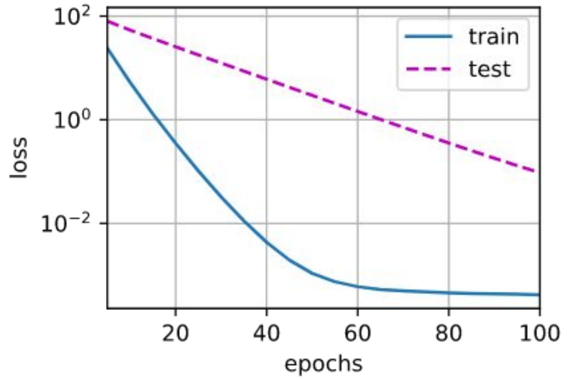
$$\mathbf{w} \leftarrow \underbrace{(1 - \eta\lambda)\,\mathbf{w}}_{\text{weight decay}} - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \mathbf{x}^{(i)} \left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right).$$

Example

Consider the simple example

$$y = 0.05 + \sum_{i=1}^{d} 0.01 x_i + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 0.01^2).$$





Weight decay /w adaptive optimizers

Weight decay /w SGD

$$\theta_{t+1} = \theta_t - \alpha \nabla f^{reg}(\theta_t) = (1-\lambda)\theta_t - \alpha \nabla f(\theta_t)$$

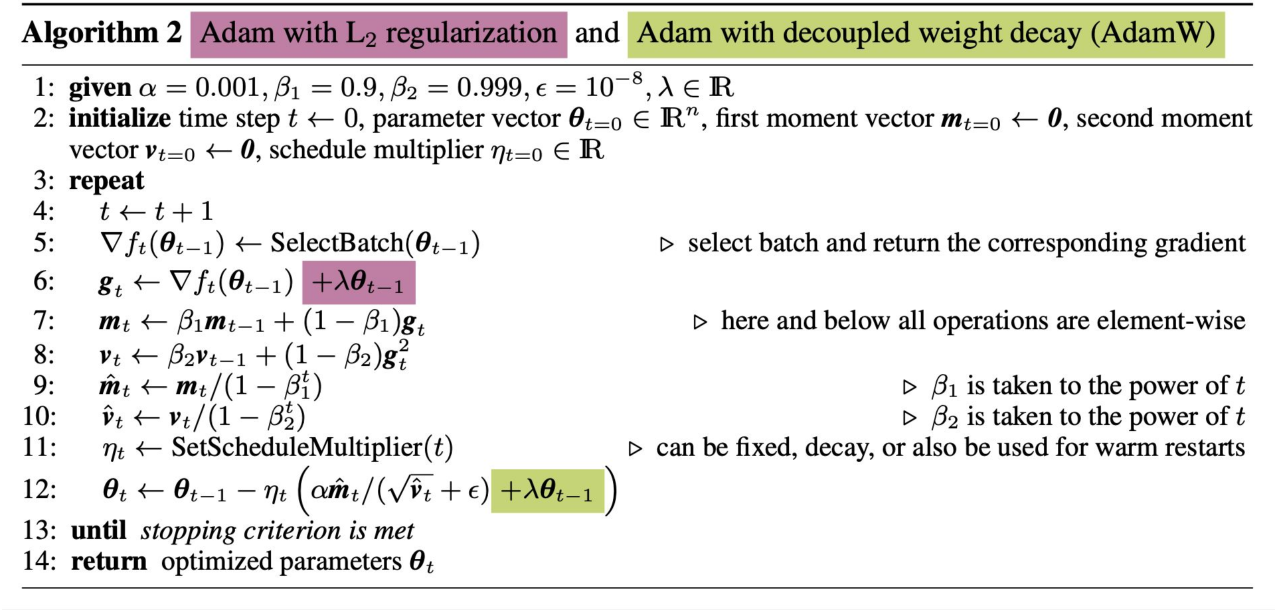
Weight decay /w Adam

$$\left.\begin{aligned} &\theta_{t+1} = \theta_t - \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1) g_t}{\sqrt{\beta_2 v_{t-1} + (1-\beta_2) g_t^2} + \varepsilon} \\ &\text{where } \; g_t = \nabla f_t(\theta_{t-1}) + \lambda \theta_{t-1} \end{aligned}\right\} \text{There is no weight decay effect!}$$

https://www.fast.ai/posts/2018-07-02-adam-weight-decay.html#adamw
https://github.com/GitYCC/machine-learning-papers-summary/blob/master/optimization-training-techniques/AdamW.md

Decoupled weight decay (AdamW)

Solution: add weight decay regularization manually



Algorithm 2 Adam with L$_2$ regularization and Adam with decoupled weight decay (AdamW)

```
1:  given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2:  initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment
    vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3:  repeat
4:      $t \leftarrow t + 1$
5:      $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$    ▷ select batch and return the corresponding gradient
6:      $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
7:      $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$    ▷ here and below all operations are element-wise
8:      $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9:      $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$    ▷ $\beta_1$ is taken to the power of $t$
10:     $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$    ▷ $\beta_2$ is taken to the power of $t$
11:     $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$    ▷ can be fixed, decay, or also be used for warm restarts
12:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left( \alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1} \right)$
13: until stopping criterion is met
14: return optimized parameters $\boldsymbol{\theta}_t$
```

Dropout

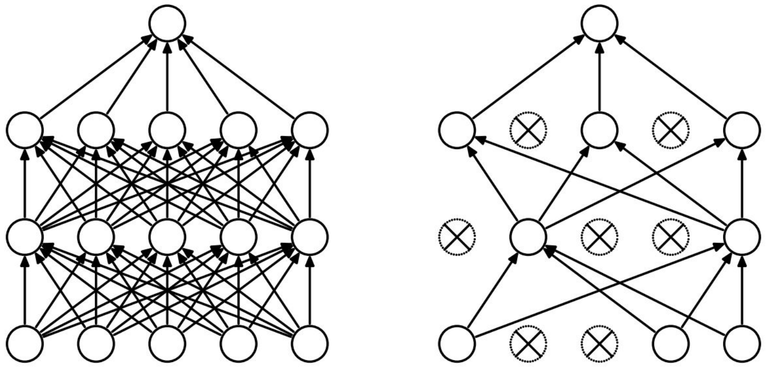
Let’s consider the following procedure:

The feed-forward operation of a standard neural network

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

With dropout, the feed-forward operation becomes

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$



(a) Standard Neural Net (b) After applying dropout.

So, in average, we have the following hidden state

$$\mathbb{E}\tilde{\mathbf{y}}^{(l)} = (1-p)y^{(l)}$$

How it works visually

How it should work on the inference?

Why does it work?

1. It learns a large ensemble of models. By doing dropout, we implicitly create a huge number of models, and from ML we know that ensembles are better than single model

2. Randomly selecting different neurons ensure that neurons are unable to learn the co-adaptations and prevent overfitting.

Recap

- Gradient descent for neural networks
    - Stochasticity
    - Momentum
    - Adaptive methods

- Weight decay
- Dropout