Regularization Techniques for Deep Learning

Materials from

- Intel Deep Learning https://www.intel.com/content/www/us/en/developer/learn/course-deep-learning.html
- Improving Deep Neural Networks https://www.deeplearning.ai/

Legal Notices and Disclaimers

Regularizing Neural Networks

We have several means by which to help “regularize” neural networks
– that is, to prevent overfitting

- Regularization penalty in cost function
- Dropout
- Early stopping "หยุดการเทรนใน point ที่เหมาะสม"
- Stochastic / Mini-batch Gradient descent (to some degree)

Penalized Cost function

- One option is to explicitly add a penalty to the loss function for having high weights.
- This is a similar approach to Ridge Regression

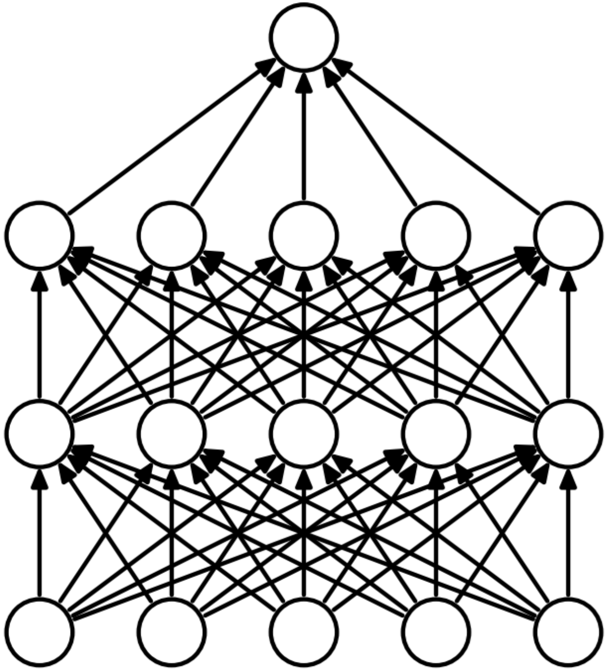
$$J = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda\sum_{j=1}^{m} W_i^2$$

- Can have an analogous expression for Categorical Cross Entropy

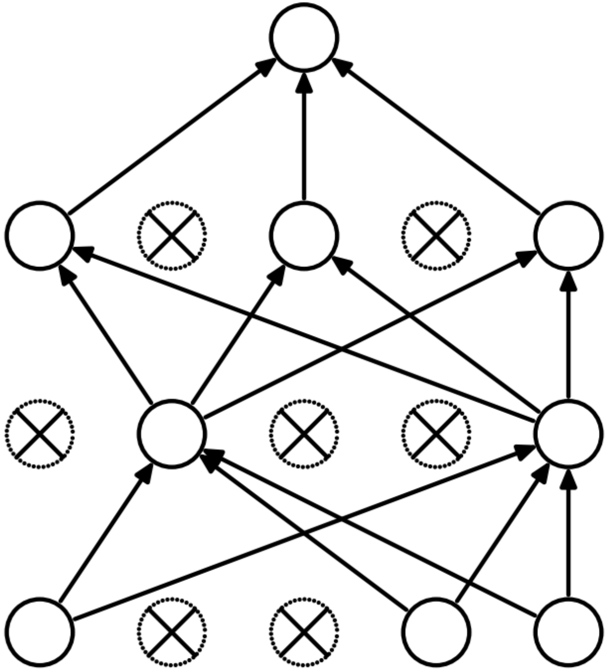
Dropout

- Dropout is a mechanism where at each training iteration (batch) we randomly remove a subset of neurons
- This prevents the neural network from relying too much on individual pathways, making it more “robust”
- At test time we “rescale” the weight of the neuron to reflect the percentage of the time it was active

Dropout - Visualization



(a) Standard Neural Net



(b) After applying dropout.

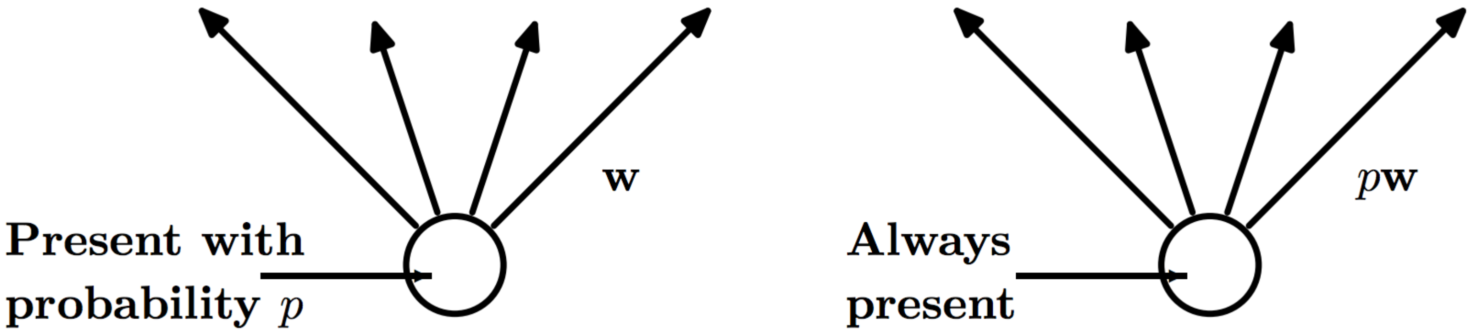
probability

p to keep : 0.4

p to drop : 0.6

Dropout - Visualization

- If the neuron was present with probability *p*, at test time we scale the outbound weights by a factor of *p*.



(a) At training time

(b) At test time

Early Stopping

"Call back func"

called when some ... was triggered

- Another, more heuristical approach to regularization is early stopping.
- This refers to choosing some rules after which to stop training.
- Example:
  - Check the validation log-loss every 10 epochs.
  - If it is higher than it was last time, stop and use the previous model (i.e. from 10 epochs previous)

Optimizers

- We have considered approaches to gradient descent which vary the number of data points involved in a step.
- However, they have all used the standard update formula:

$$\mathrm{W} := W - \alpha \cdot \nabla J$$

- There are several variants to updating the weights which give better performance in practice.
- These successive “tweaks” each attempt to improve on the previous idea.
- The resulting (often complicated) methods are referred to as “optimizers”.

Exponentially Weighted Averages

$\theta_1 = 40°F$ $4°C \leftarrow$

$\theta_2 = 49°F$ $9°C$

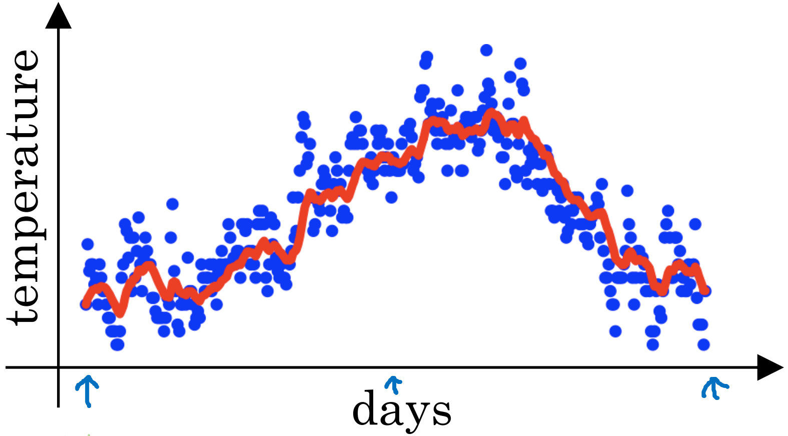
$\theta_3 = 45°F$ $\vdots$

$\vdots$

$\theta_{180} = 60°F$ $15°C$

$\theta_{181} = 56°F$ $\vdots$

$\vdots$



$V_0 = 0$

$V_1 = \underbrace{0.9}_{\text{beta}} V_0 + \underbrace{0.1}_{\text{1-beta}} \underbrace{\theta_1}_{\text{data}}$

$V_2 = 0.9 V_1 + 0.1 \theta_2$

$V_3 = 0.9 V_2 + 0.1 \theta_3$

$\vdots$

$\boxed{V_t = 0.9 V_{t-1} + 0.1 \theta_t}$

Andrew Ng

Exponentially weighted averages

moving

$$V_t = \beta V_{t-1} + (1-\beta)\theta_t \leftarrow$$

$\beta = 0.9$: ≈ 10 days' temperature.

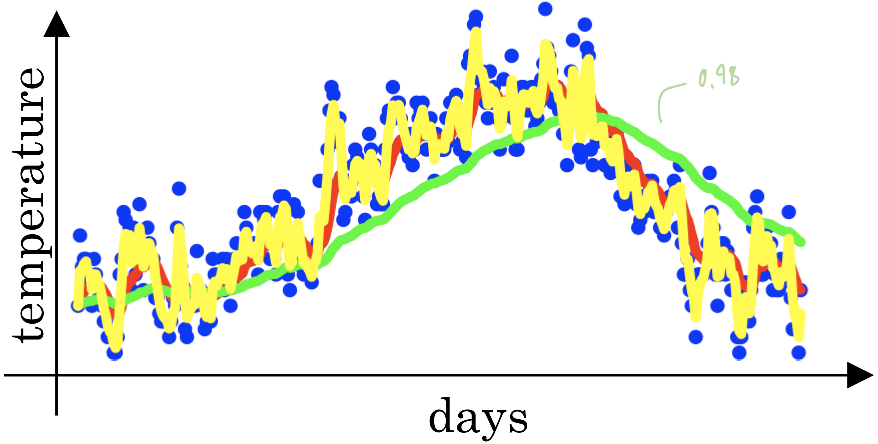
$\beta = 0.98$: ≈ 50 days

$\beta = 0.5$: ≈ 2 days

V_t as approximately average over

$\rightarrow \approx \frac{1}{1-\beta}$ days' temperature.

$$\frac{1}{1-0.98} = 50$$



Andrew Ng

Momentum

- Idea, only change direction by a little bit each time.
- Keeps a "running average" of the step directions, smoothing out the variation of the individual points.

MOMENTUM!!

previous one

gradient

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot \nabla J$$

average value over time

learning rate

$$\mathrm{W} = W - \alpha \cdot v_t$$

- Here, β is referred to as the "momentum". It is generally given a value <1

Momentum

- Idea, only change direction by a little bit each time.
- Keeps a "running average" of the step directions, smoothing out the variation of the individual points.

default : 0.9

$$v_t = \beta \cdot v_{t-1} + \cancel{(1-\beta)} \cdot \nabla J$$

often omitted in literatures

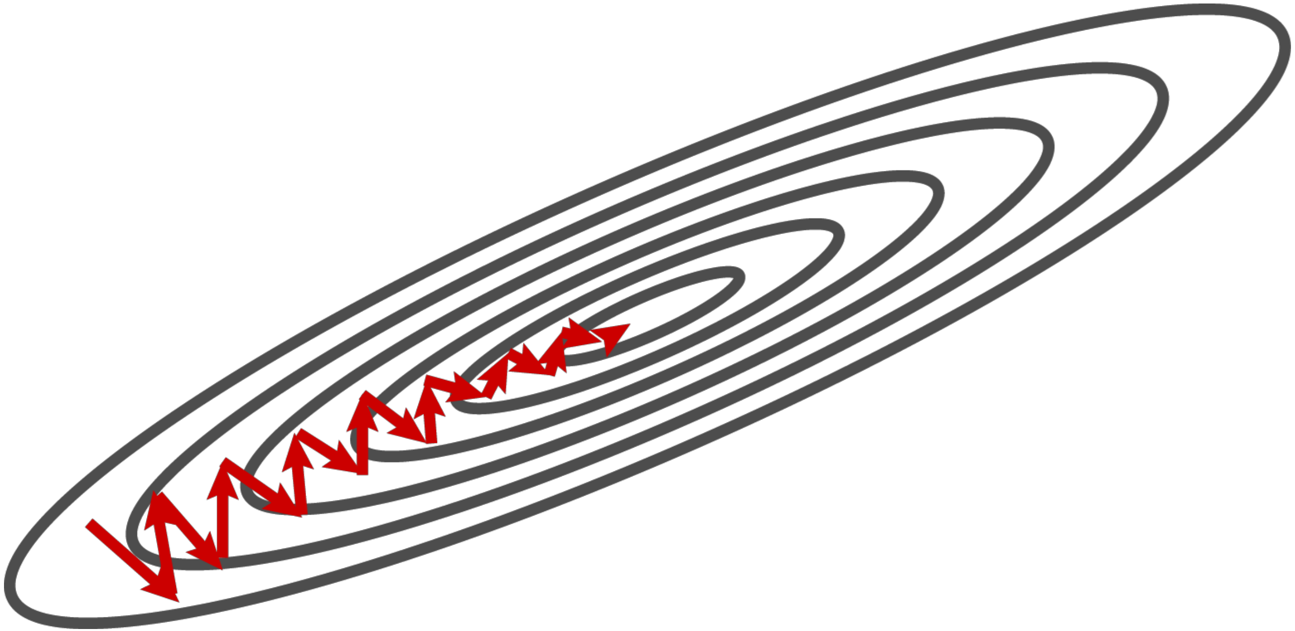
(learning rate

$$W = W - \alpha \cdot v_t$$

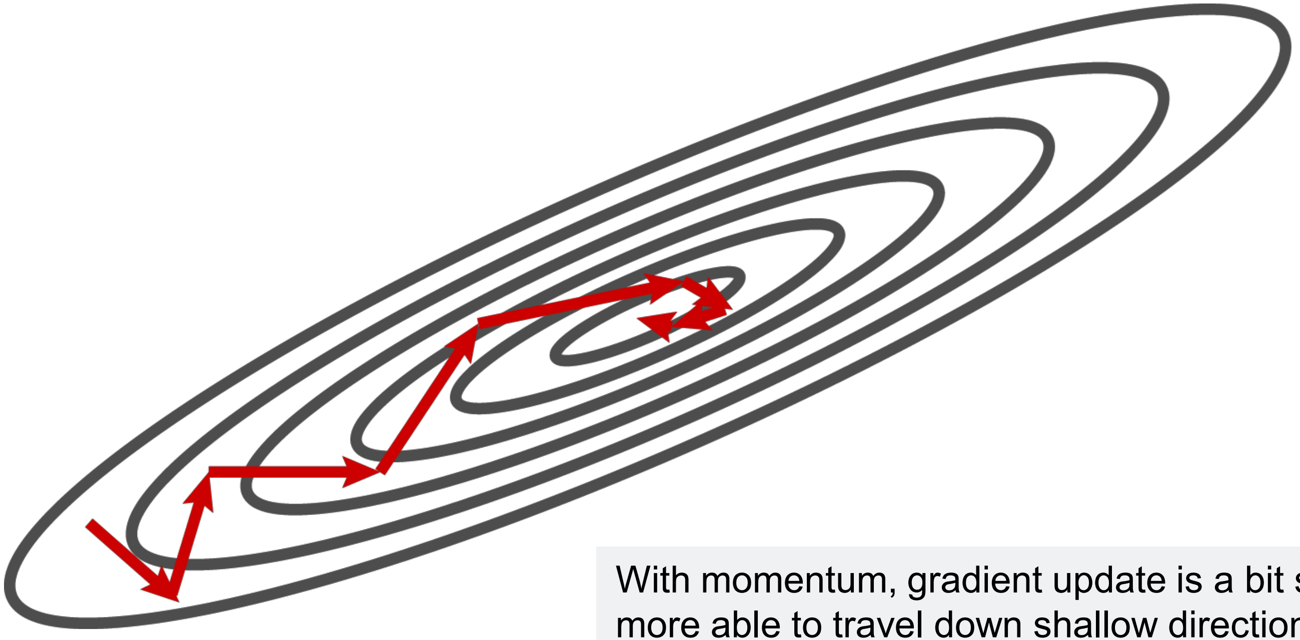
not general in some paper

- Here, β is referred to as the "momentum". It is generally given a value <1

Gradient Descent vs Momentum



Gradient Descent vs Momentum



With momentum, gradient update is a bit smoother and more able to travel down shallow directions.

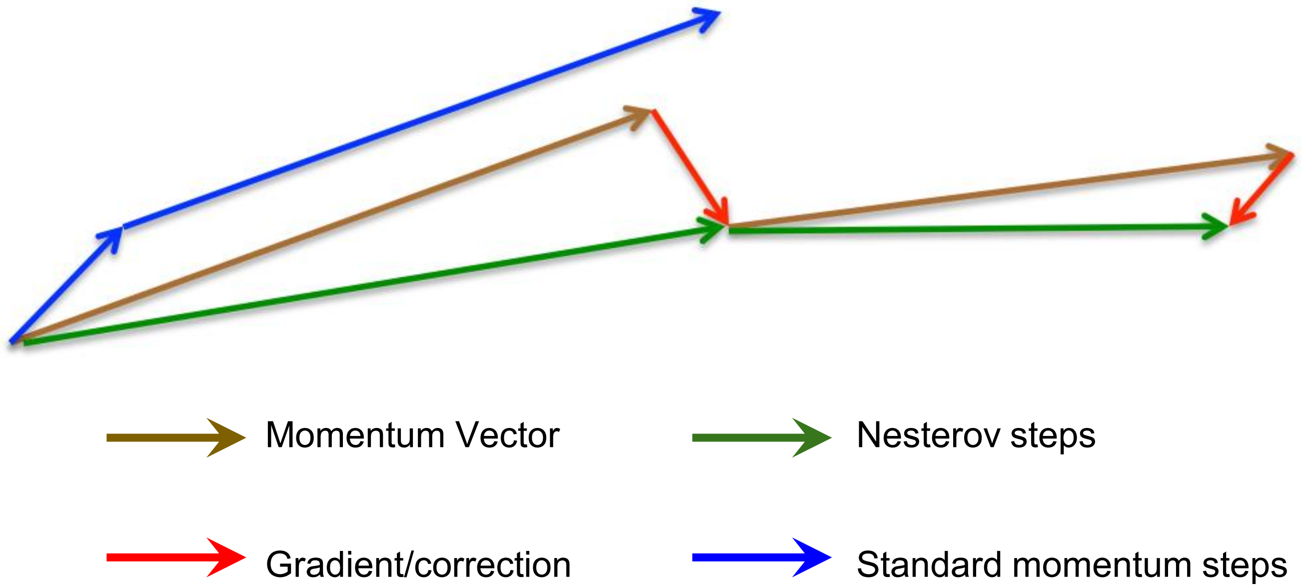
Nesterov Momentum

- Idea: Control "overshooting" by looking ahead.
- Apply gradient only to the "non-momentum" component.

$$v_t = \beta \cdot v_{t-1} + \alpha \nabla J(\overbrace{W - \beta \cdot v_{t-1}}^{\text{lookahead}})$$

$$\mathrm{W} = W - v_t$$

Nesterov Momentum



AdaGrad (Adaptive Gradient Optimizer)

- Idea: scale the update for each weight separately.
- Keep running sum of previous updates
- Divide new updates by factor of previous sum

$$W = W - \frac{\alpha}{\sqrt{G_t + \epsilon}} \nabla J \qquad\qquad G_t = G_{t-1} + (\nabla J)^2$$

dynamic learning rate

high -> low but it'll be too low at further point

- Instead of using constant learning rate, the learning rate is then divided by the square root of the sum of each component separately.
- As a result of this normalization, those weights that are associated with high gradients will have their learning rates suppressed more.
- This aggressive decay of the rate, however, turns out to be too strong.

RMSProp (Root Mean Square Propagation)

- Quite similar to AdaGrad.
- Rather than using the sum of previous gradients, decay older gradients more than more recent ones.
- More adaptive to recent updates
- It provides an adaptive learning rate which suppresses learning rates for weights with large frequent gradient updates.

$\frac{\partial J}{\partial W}$

$$S_{dW} = \beta\, S_{dW_{prev}} + (1-\beta)(dW)^2$$

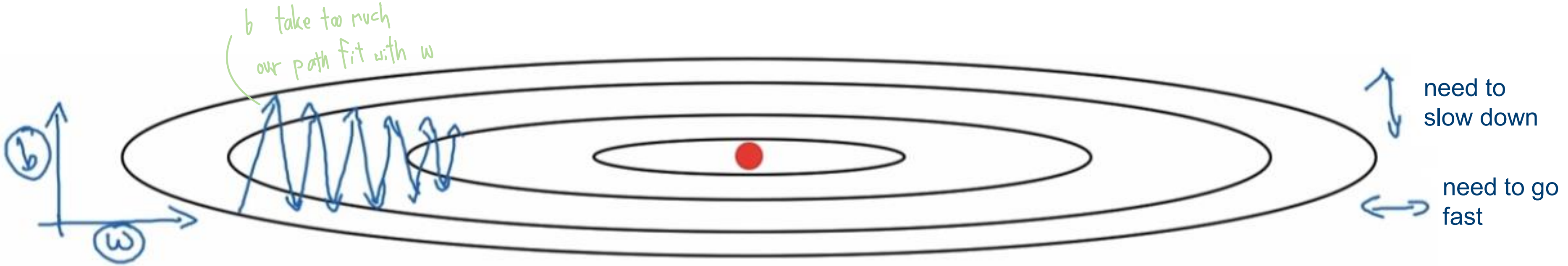
$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \epsilon}$$

prevent divided by 0

RMSProp (Case Study)

$$S_{dW} = \beta\, S_{dW_{prev}} + (1-\beta)(dW)^2 \qquad S_{db} = \beta\, S_{db_{prev}} + (1-\beta)(db)^2$$

$$W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \epsilon} \qquad b = b - \alpha \frac{db}{\sqrt{S_{db}} + \epsilon}$$



Adam (Adaptive Moment Estimation)

- Idea: blending between momentum and RMSprop.
- For iteration t:

$$\mathrm{V_{dW}} = \beta_1 \mathrm{V_{dW}}_{prev} + (1-\beta_1)\mathrm{dW} \qquad \mathrm{S_{dW}} = \beta_2 \mathrm{S_{dW}}_{prev} + (1-\beta_2)(\mathrm{dW})^2$$

$$\hat{\mathrm{V}}_{\mathrm{dW}} = \frac{\mathrm{V_{dW}}}{1-\beta_1^t} \qquad \hat{\mathrm{S}}_{\mathrm{dW}} = \frac{\mathrm{S_{dW}}}{1-\beta_2^t}$$

$$W := W - \alpha \frac{\hat{\mathrm{V}}_{\mathrm{dW}}}{\sqrt{\hat{\mathrm{S}}_{\mathrm{dW}}} + \epsilon}$$

Which one should I use?!

- RMSProp and Adam seem to be quite popular now.
- Difficult to predict in advance which will be best for a particular problem.
- Still an active area of inquiry.

Software