Regularization Techniques for Deep Learning

Legal Notices and Disclaimers

This presentation is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

This sample source code is released under the Intel Sample Source Code License Agreement.

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
- 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

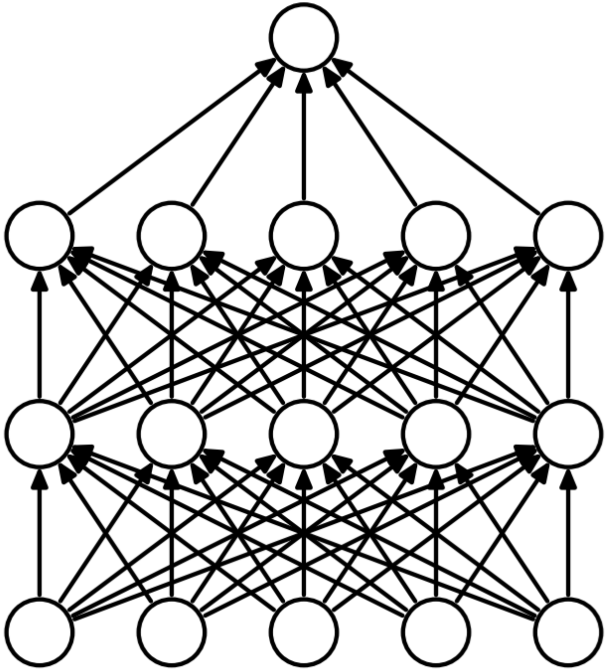
Dropout

- Dropout is a mechanism where at each training iteration (batch) we randomly remove a subset of neurons

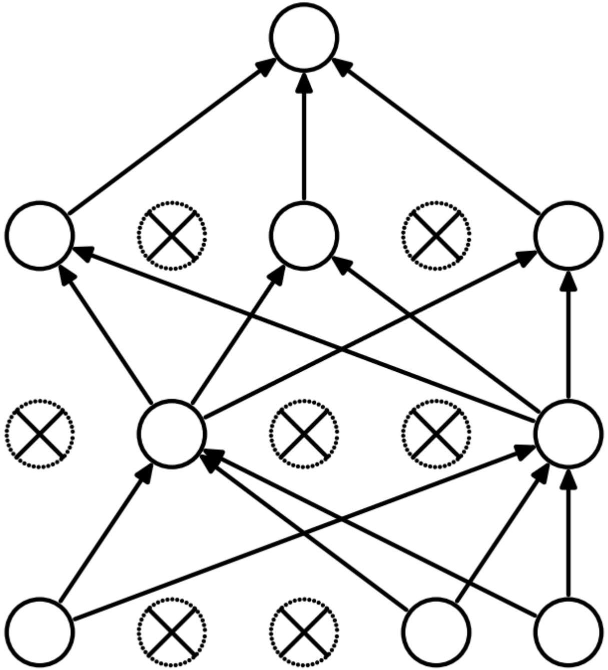
- This prevents the neural network from relying too much on individual pathways, making it more “robust”

- At test time we “rescale” the weight of the neuron to reflect the percentage of the time it was active

Dropout - Visualization



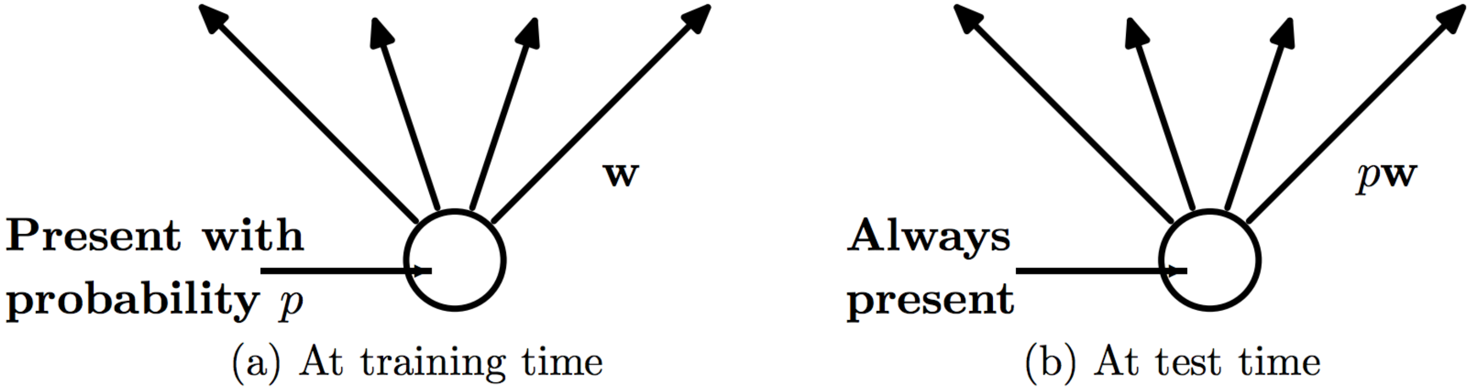
(a) Standard Neural Net



(b) After applying dropout.

Dropout - Visualization

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



Early Stopping

- 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:

$$\mathrm{W} \coloneqq W - \alpha \cdot \nabla J$$

- 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”.

Momentum

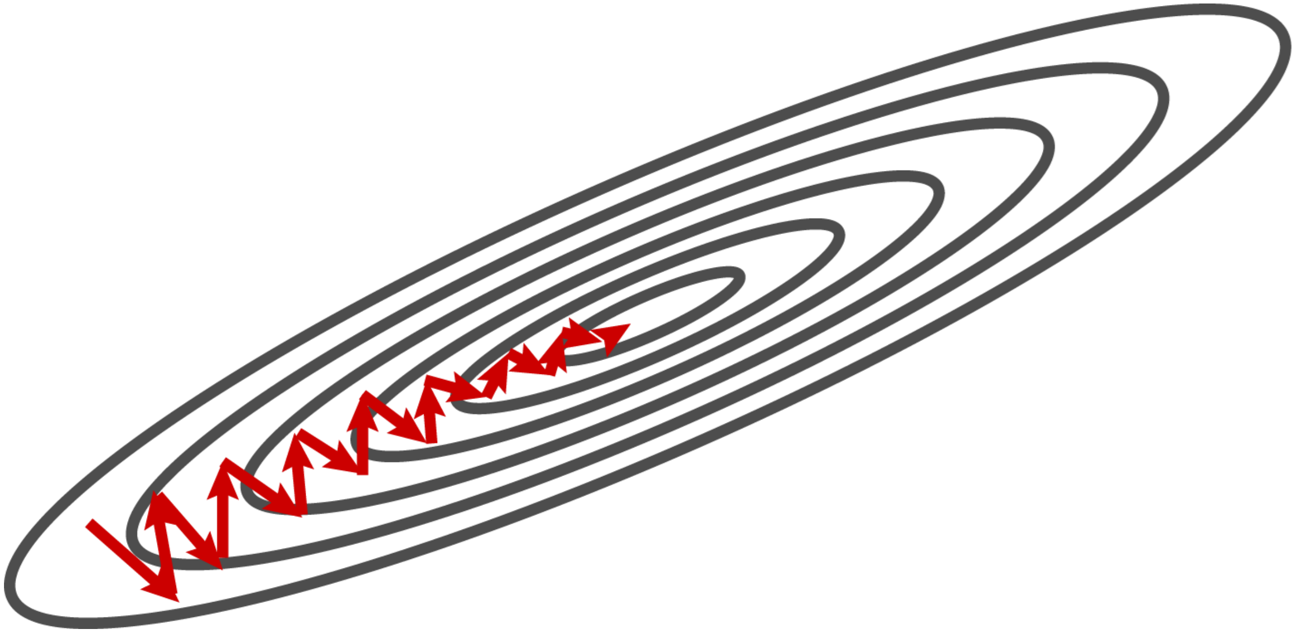
- Idea, only change direction by a little bit each time.
- Keeps a "running average" of the step directions, smoothing out the variation of the individual points.

$$v_{\mathrm{t}} \coloneqq \eta \cdot v_{t-1} - \alpha \cdot \nabla J$$

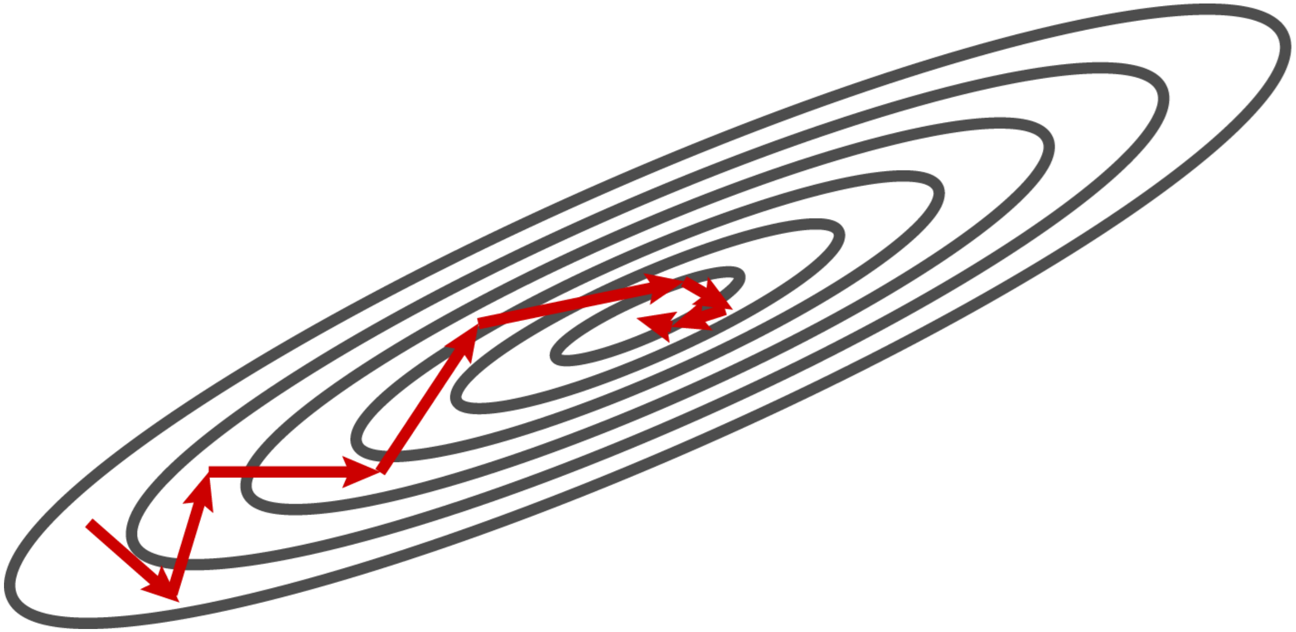
$$\mathrm{W} \coloneqq W - v_t$$

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

Gradient Descent vs Momentum



Gradient Descent vs Momentum



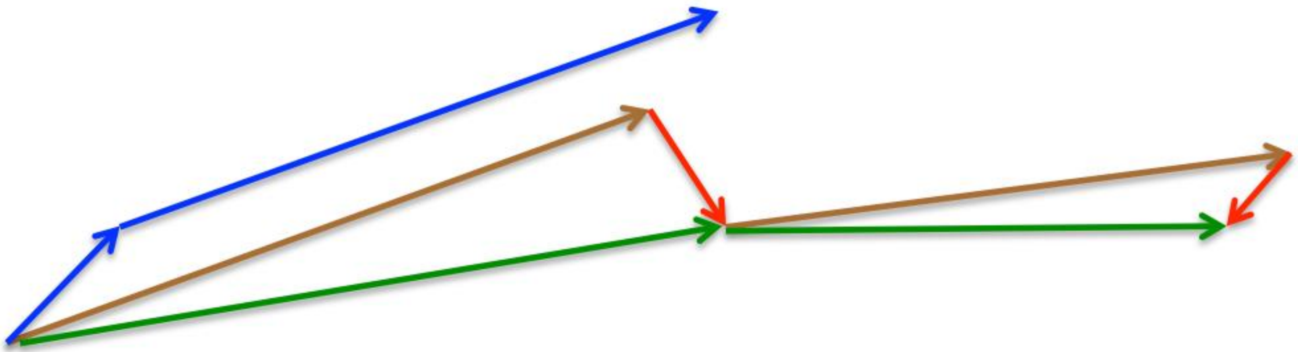
Nesterov Momentum

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

$$v_t = \eta \cdot v_{t-1} - \alpha \cdot \nabla(J - \eta \cdot v_{t-1})$$

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

Nesterov Momentum



Momentum Vector

Nesterov steps

Gradient/correction

Standard momentum steps

AdaGrad

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

$$W := W - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot \nabla J$$

RMSProp

- 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

Adam

- Idea: use both first-order and second-order change information and decay both over time.

$$\mathrm{m_t} = \beta_1 \mathrm{m_{t-1}} + (1 - \beta_1)\nabla \mathrm{J}$$

$$\mathrm{v_t} = \beta_2 \mathrm{v_{t-1}} + (1 - \beta_2)\nabla \mathrm{J}$$

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$W := W - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \odot \widehat{m}_t$$

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