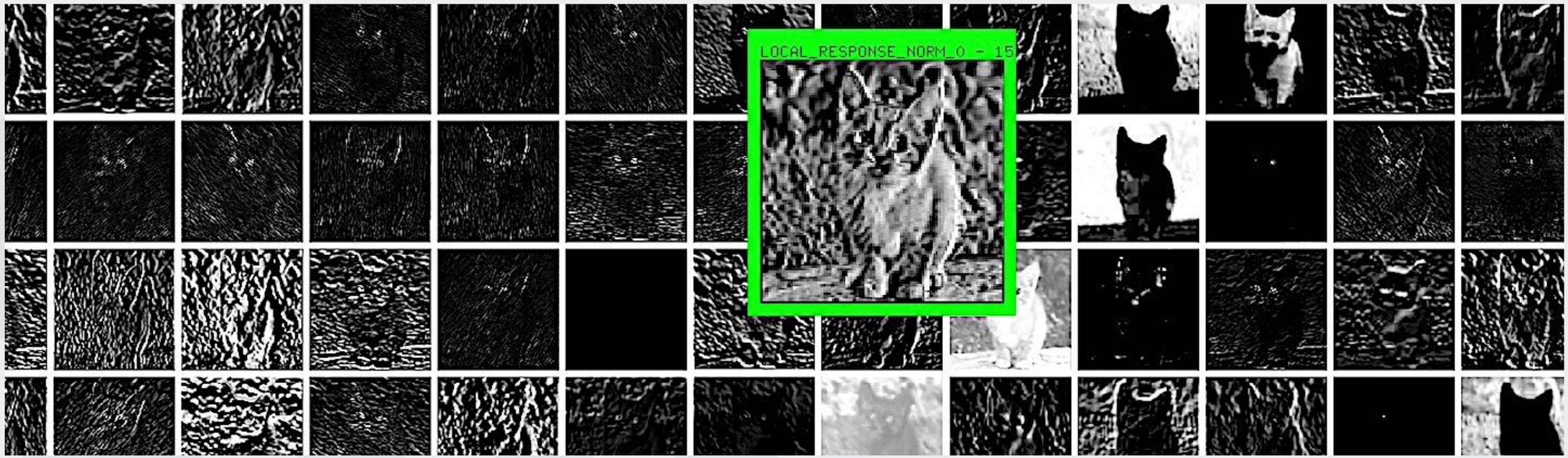
Practical Creativity with Machine Learning

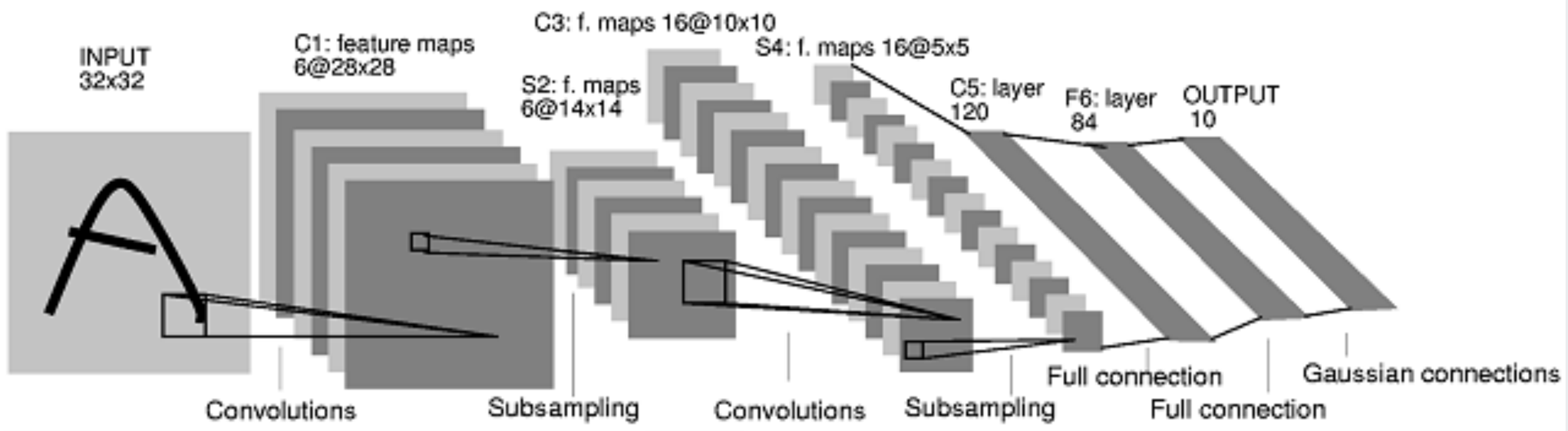
Session 6: Optimization & scaling up

genekogan.com
@genekogan

ml4a.github.io
@ml4a_

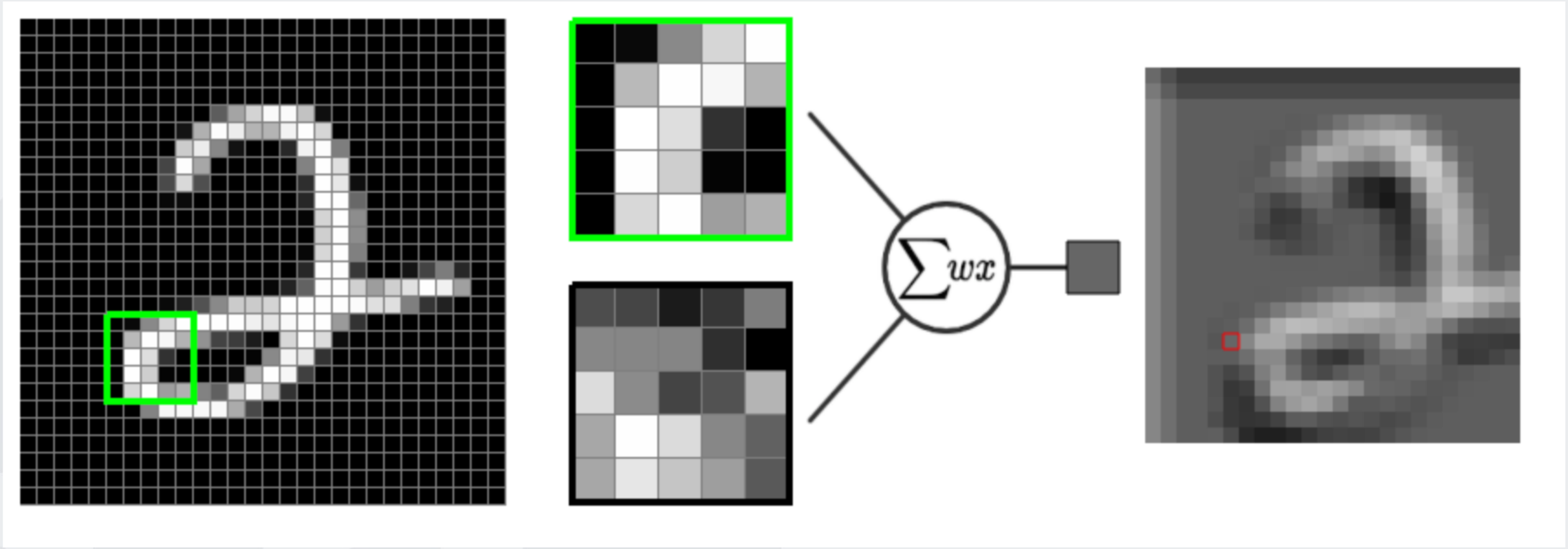


Lenet (LeCun et al, 1998)
http://yann.lecun.com/exdb/lenet/



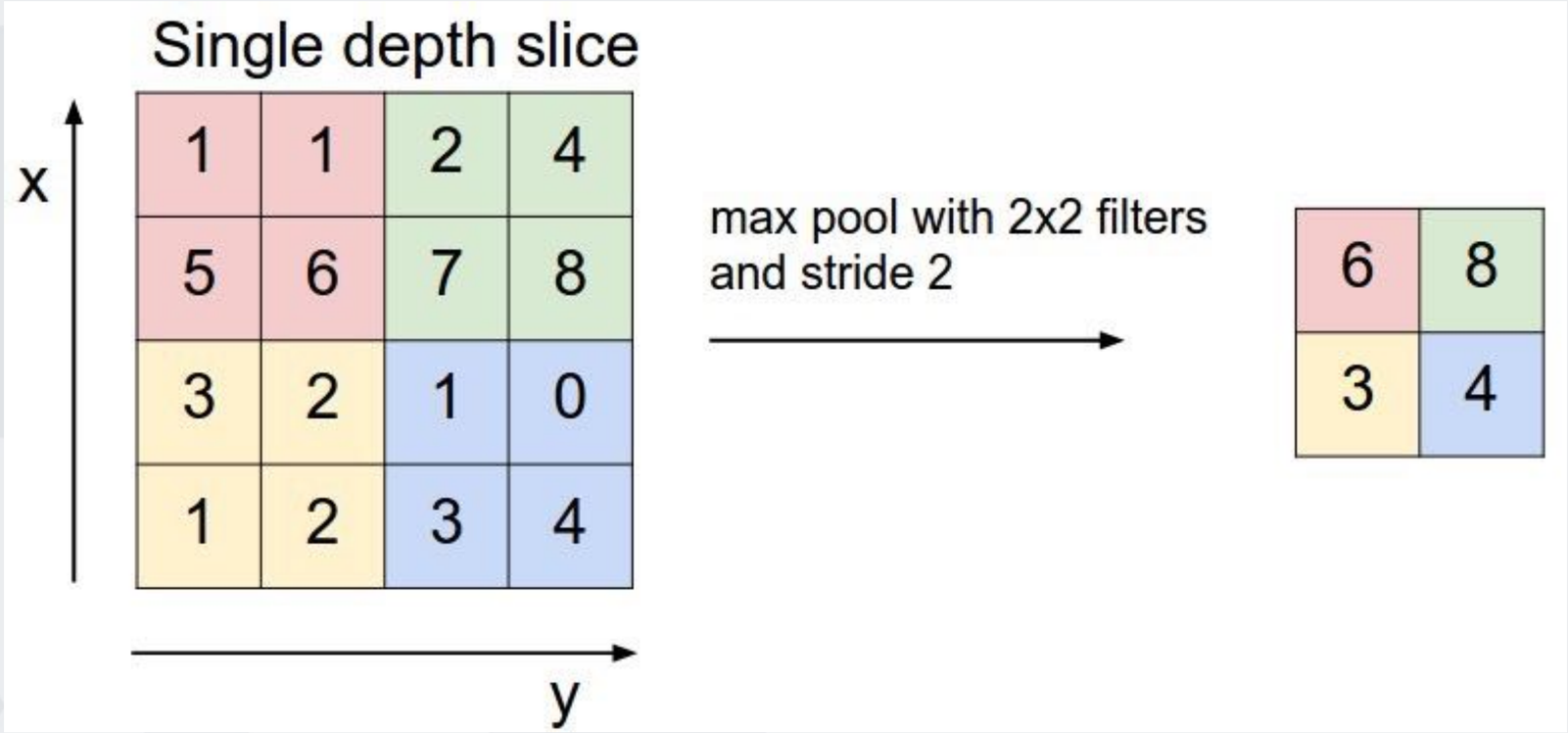
Convolution

ml4a.github.io/demos/convolution/



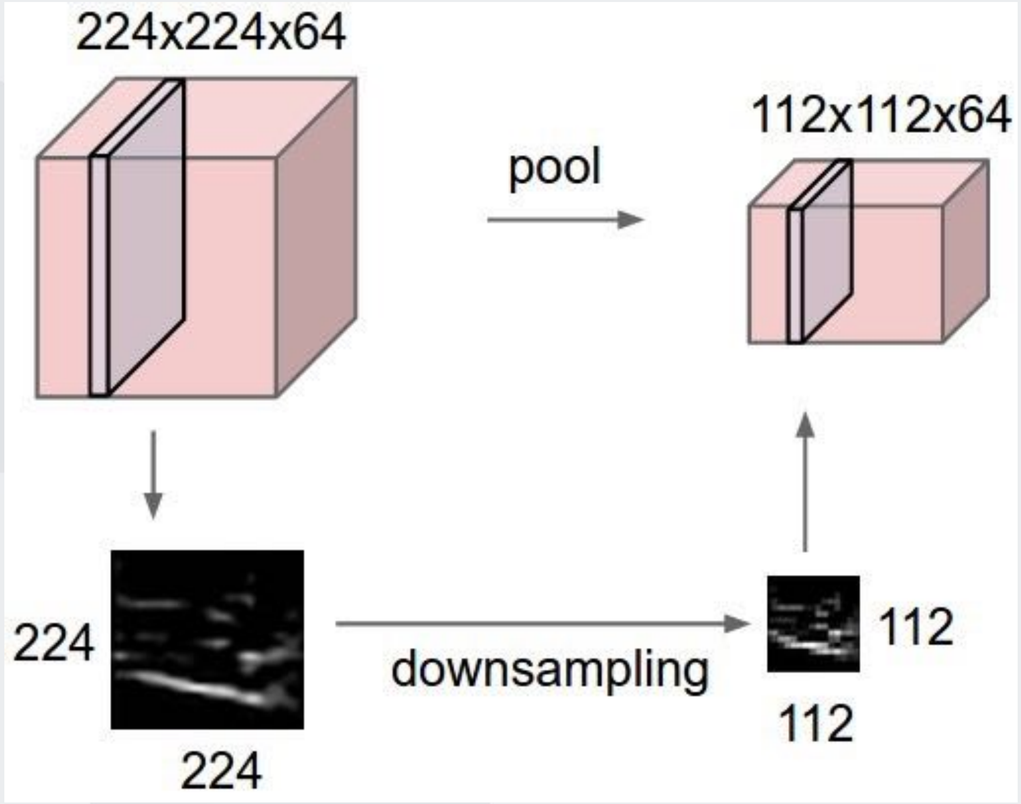
Pooling layers

cs231n.github.io/convolutional-networks/



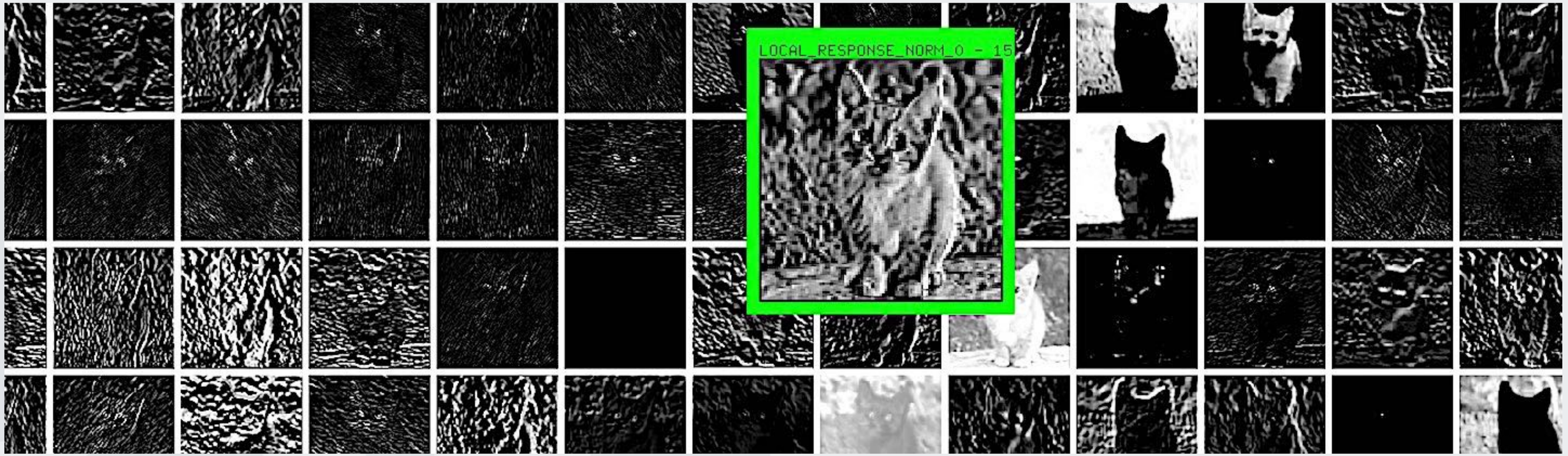
Pooling layers

cs231n.github.io/convolutional-networks/



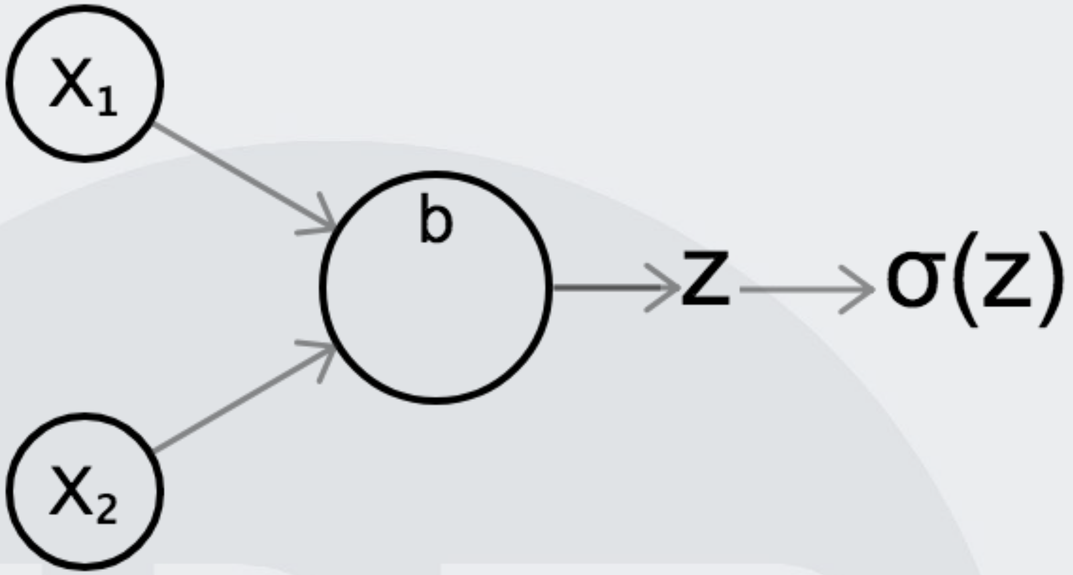
Convolutional neural network visualizer

ml4a.github.io/guides/ConvnetViewer/

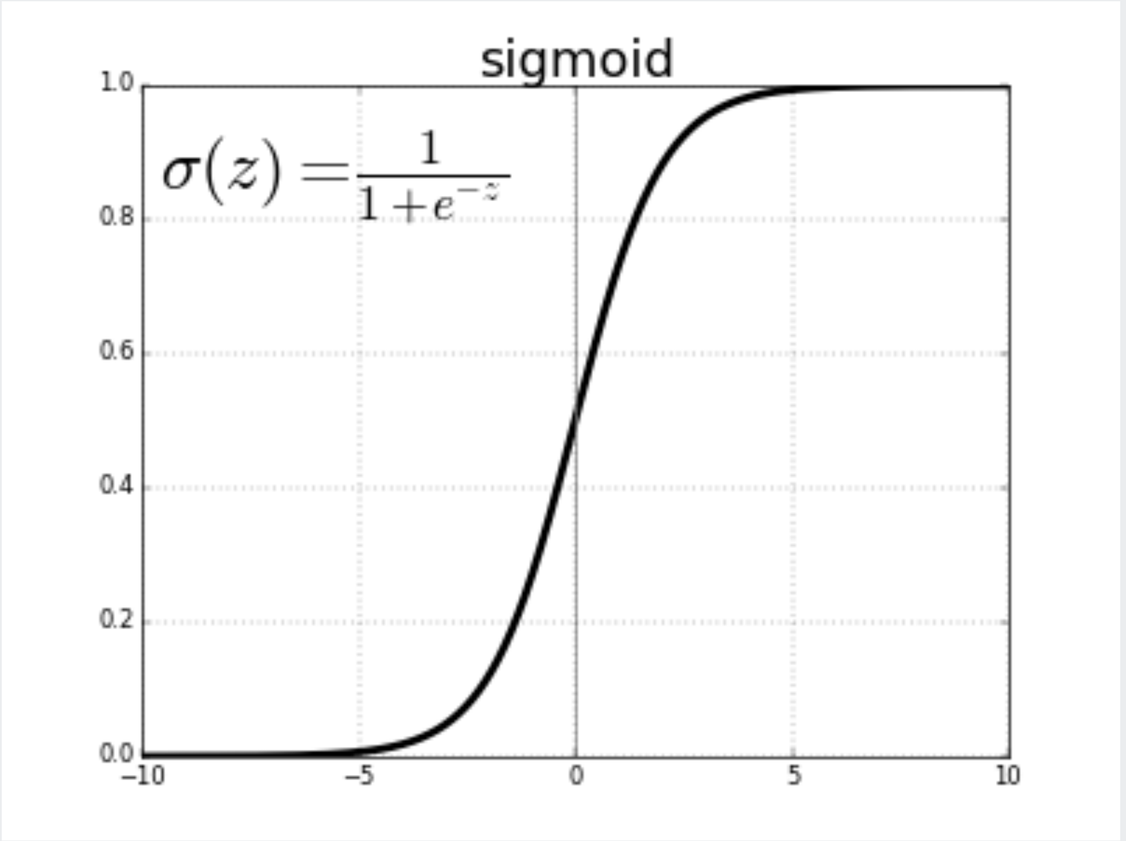


Activation functions

- Sigmoid activation functions are not good in deep neural networks because of the "vanishing gradient problem"

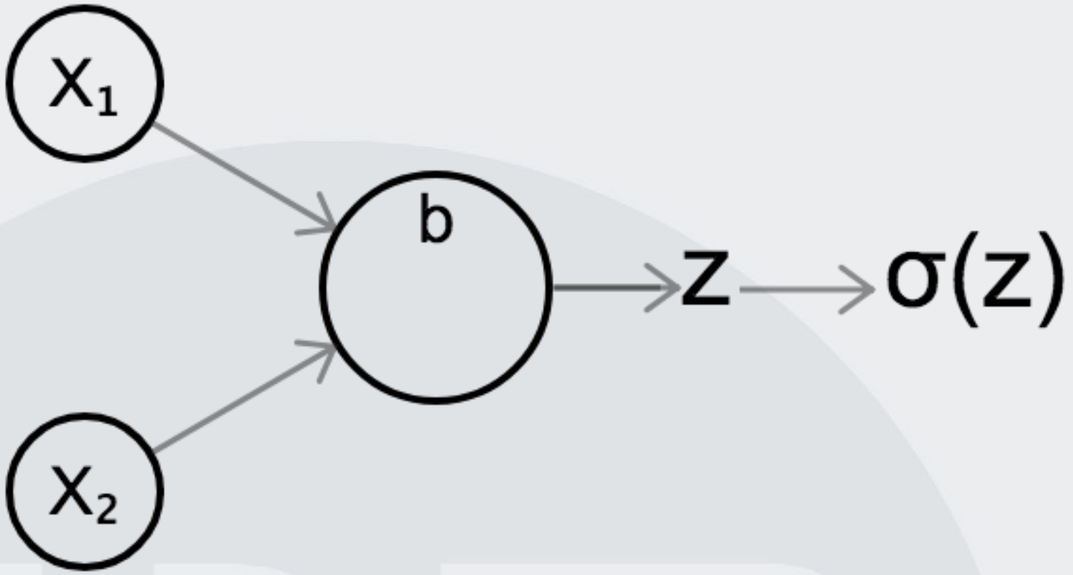


$$z = w_1 x_1 + w_2 x_2 + b$$
$$f(X) = \sigma(z)$$

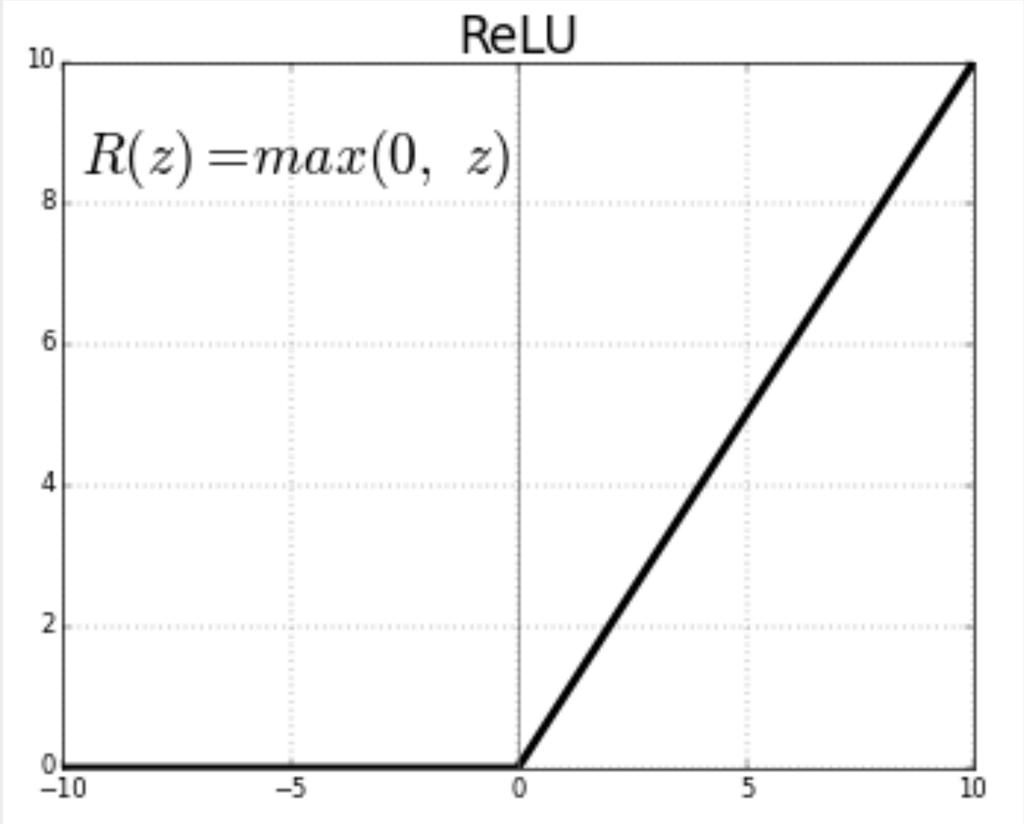


ReLU

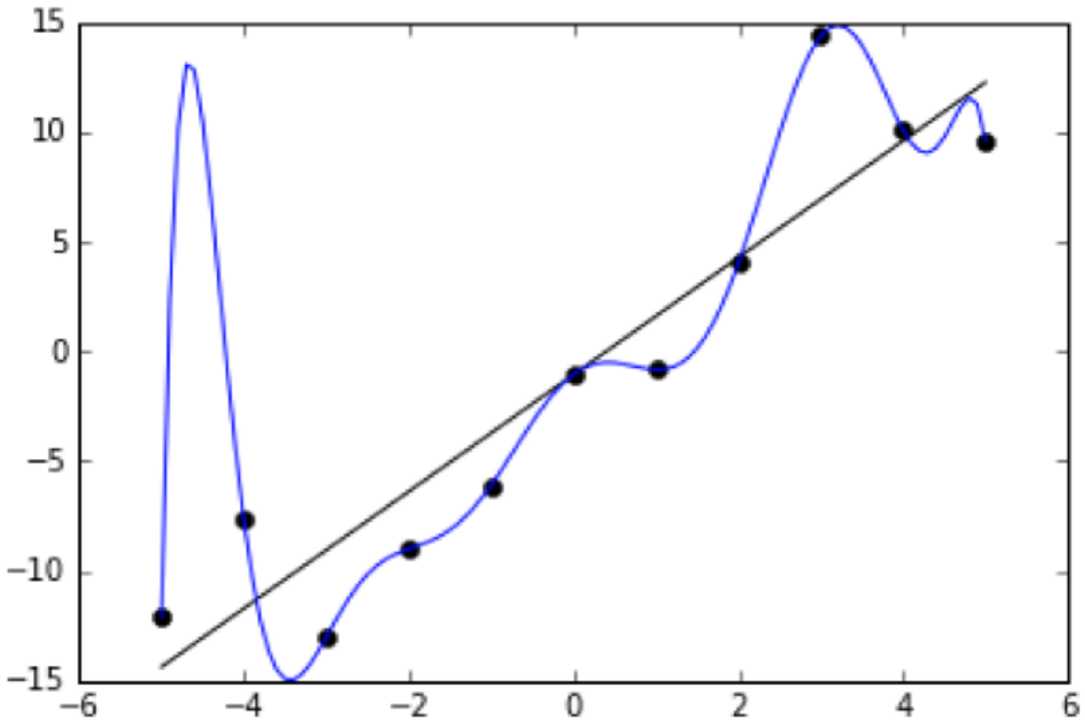
- Geoffrey Hinton et al introduced ReLU (rectified linear units) in 2011
- There are some variants



$$z = w_1 x_1 + w_2 x_2 + b$$
$$f(X) = R(z)$$



Overfitting



An example of overfitting. The straight line is simple and roughly captures the data points with some error. The curvy line has 0 error but is very complex and likely does not generalize well.

Source: Wikipedia.

Regularization - L2

- Normal, unregularized mean-squared error loss function (can be any loss function)

$$J = \frac{1}{n}\sum_{i}(y_i - f(x_i))^2$$

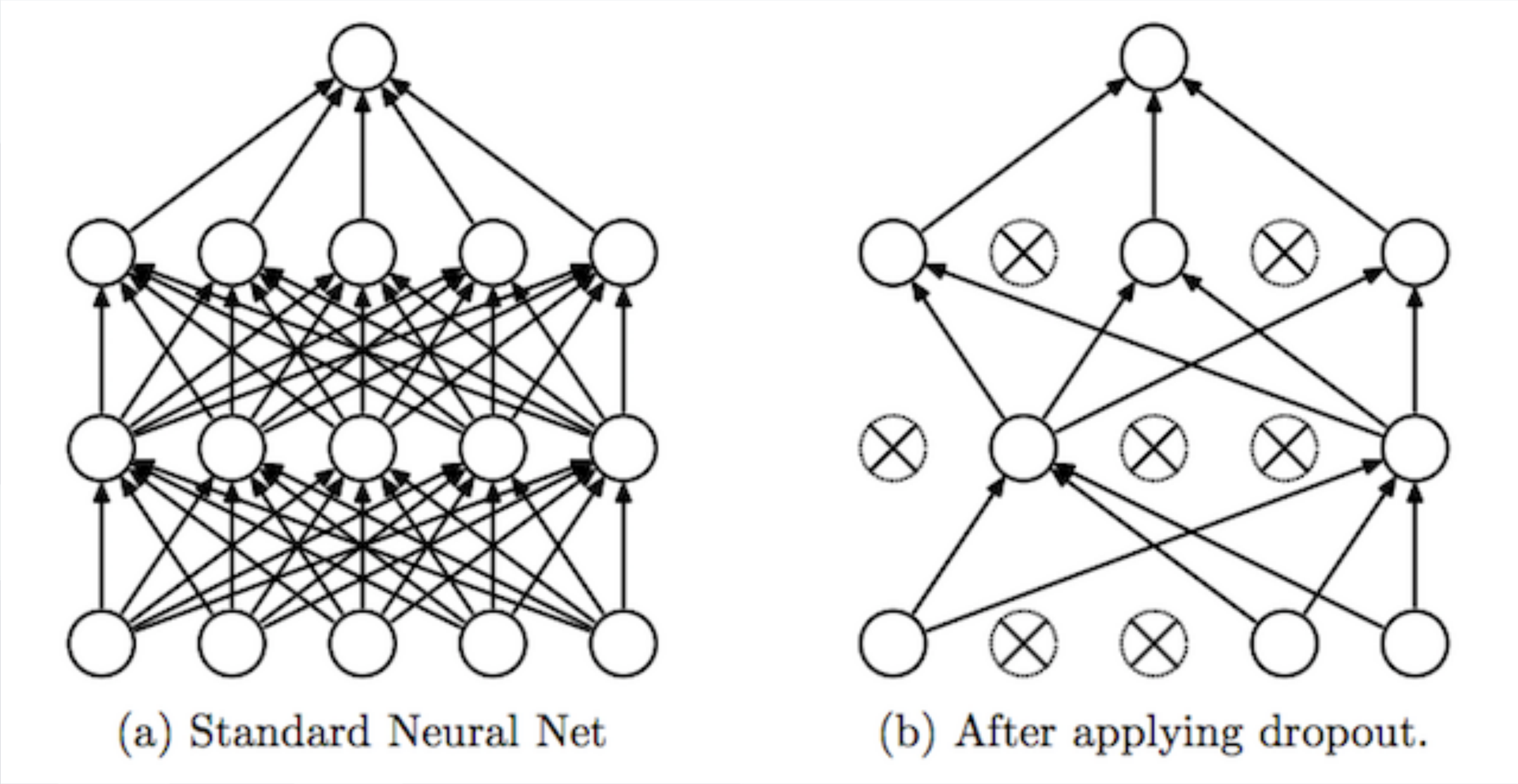
- We can penalize large weights by adding the squares of the weights to our loss term.
- Incentivize the network to learn small, evenly distributed weights

$$R(f) = \frac{1}{2}\lambda\sum w^2$$

$$J = \frac{1}{n}\sum_{i}(y_i - f(x_i))^2 + R(f)$$

Regularization - Dropout

- Geoffrey Hinton et al introduced Dropout in 2014
- During training, at some layers, randomly select some of the neurons and don't update their weights

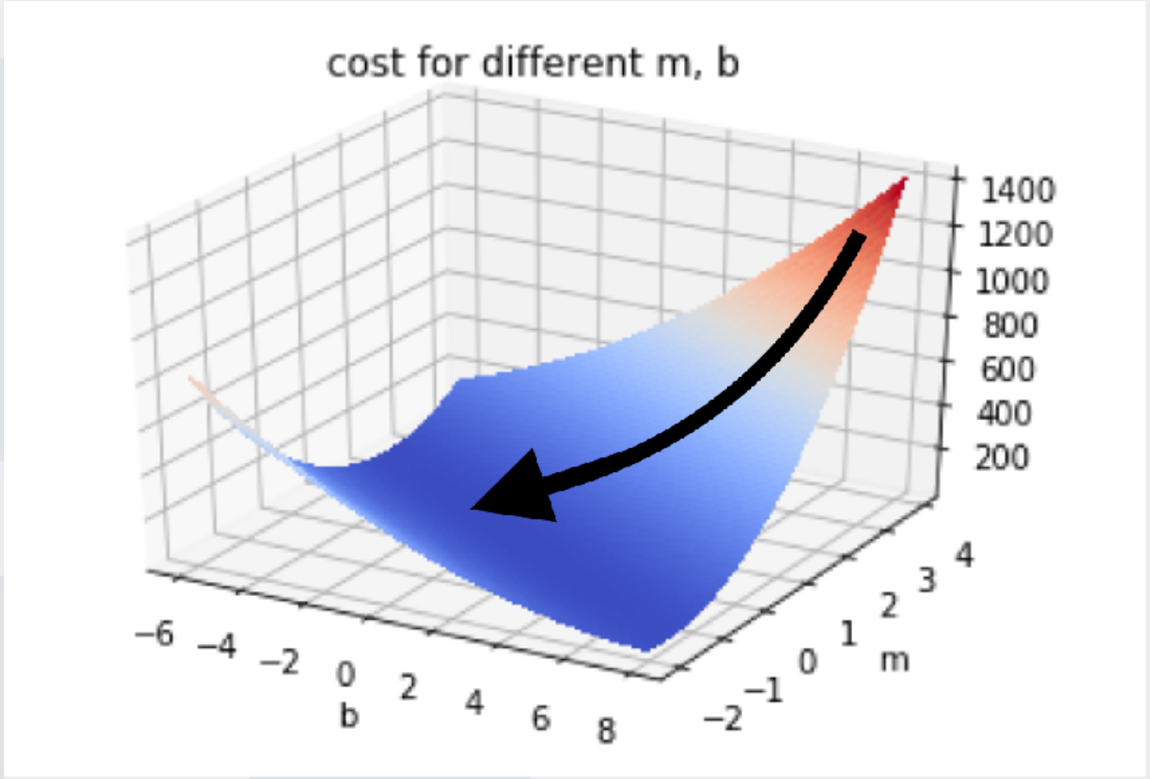


(a) Standard Neural Net

(b) After applying dropout.

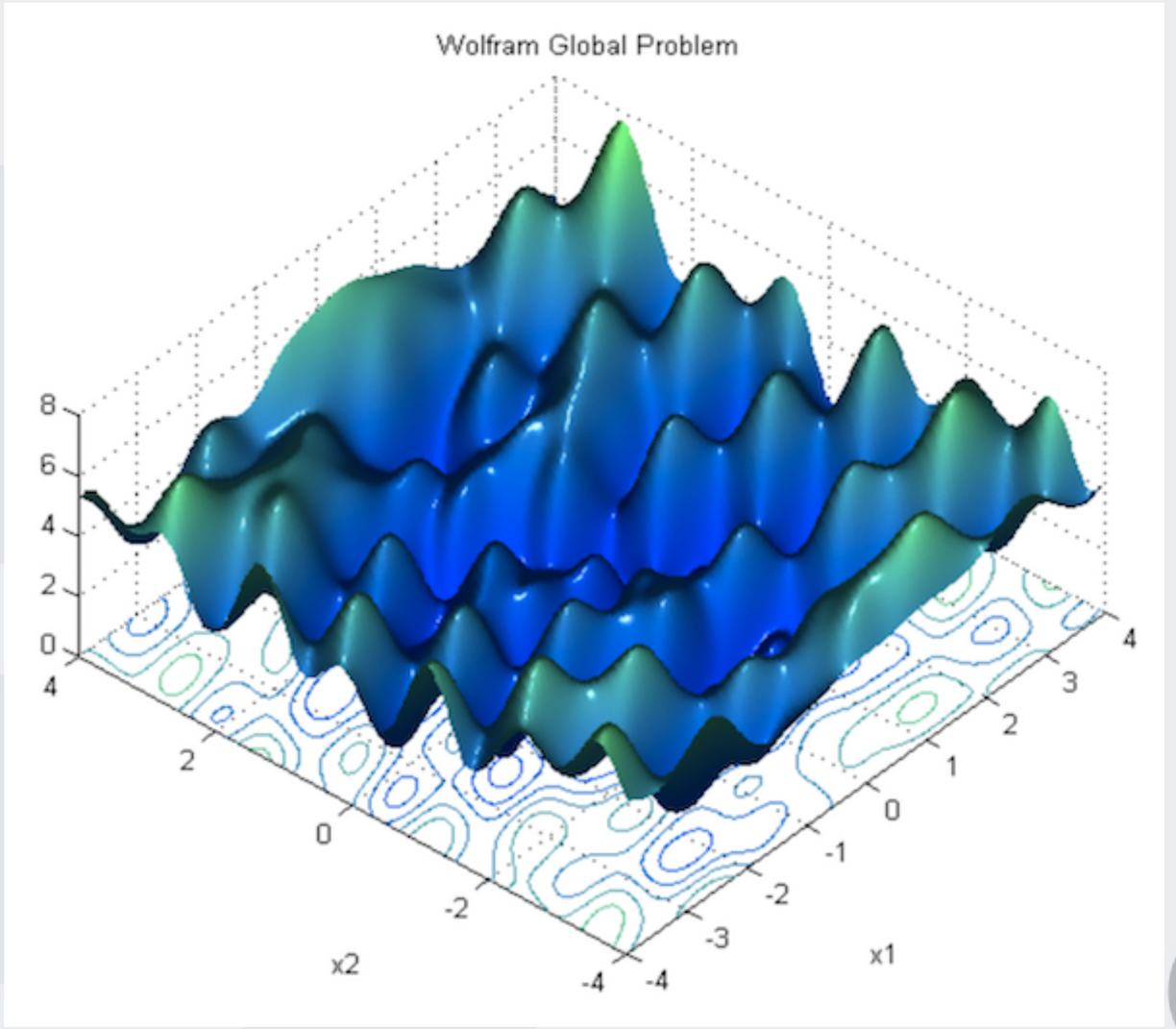
Gradient descent has no problems for linear regression

- The loss function is convex, or “bowl-shaped”. Gradient descent is guaranteed to find the bottom.



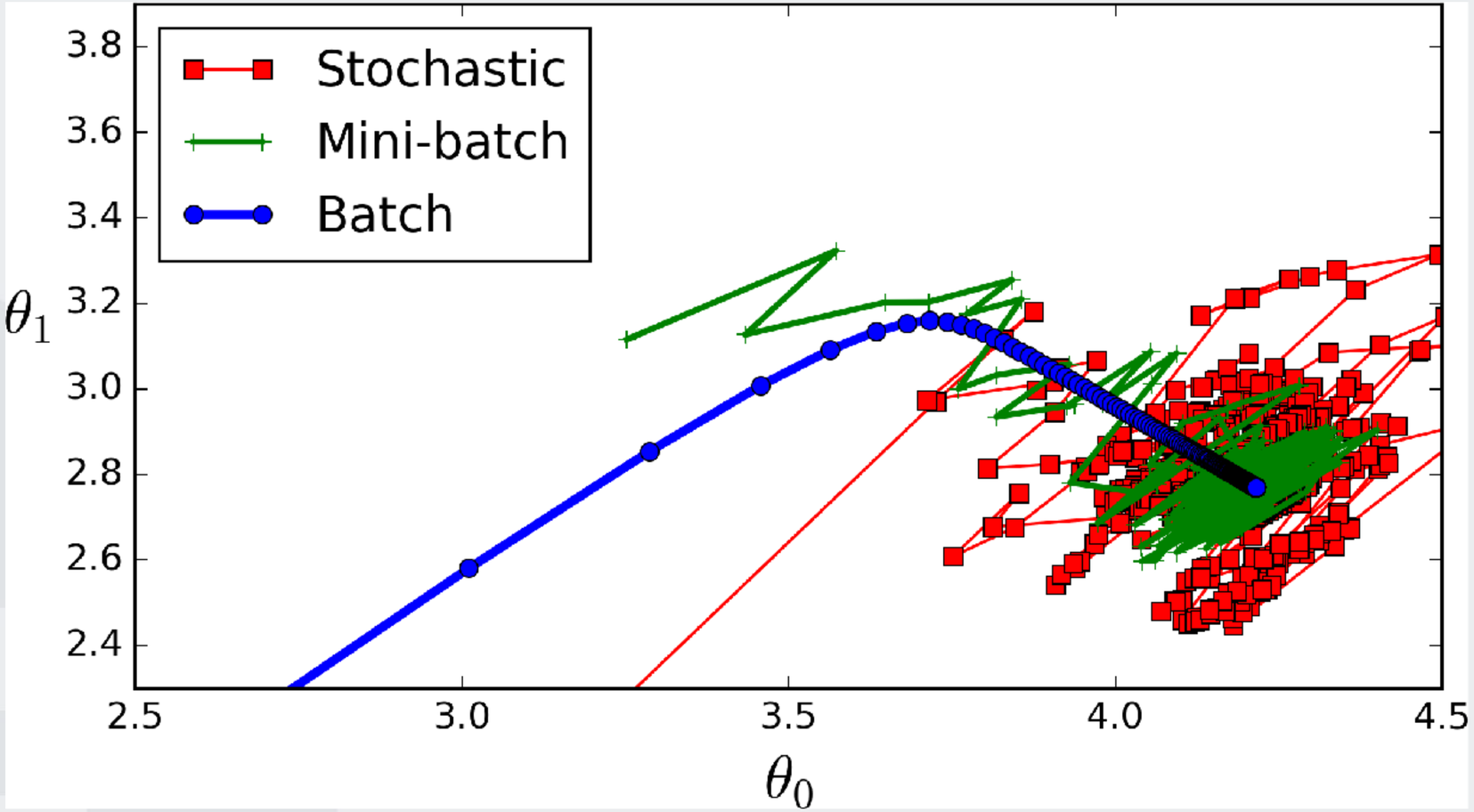
Problem with gradient descent

- What we actually have: local minima and saddle points



Flavors of gradient descent

- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent



- Source: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network

Momentum

- Standard weight update with batch, mini-batch, or SGD

$$W_t := W_t - \alpha \nabla J(W_t)$$

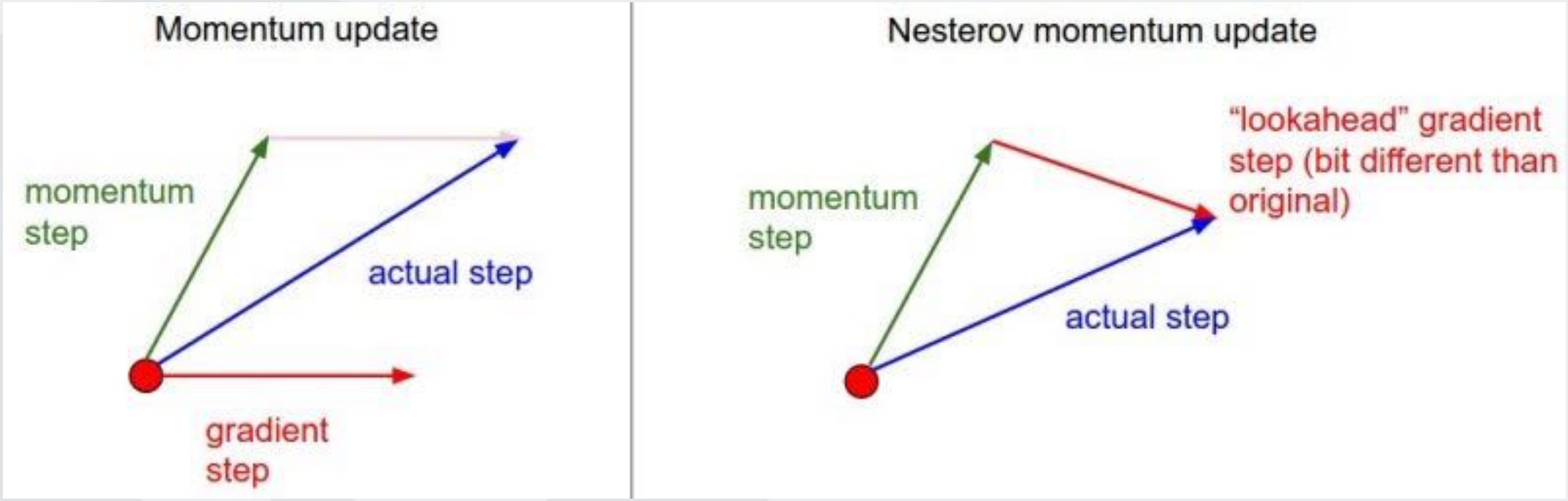
- With momentum, we “ease” the velocity of the weight update, giving past updates some additional life.
- We might be able to “roll through” saddle points and local minima.

$$z_t := \beta z_{t-1} + \nabla J(W_{t-1})$$

$$W_t := W_{t-1} - \alpha z_t$$

Nesterov Momentum

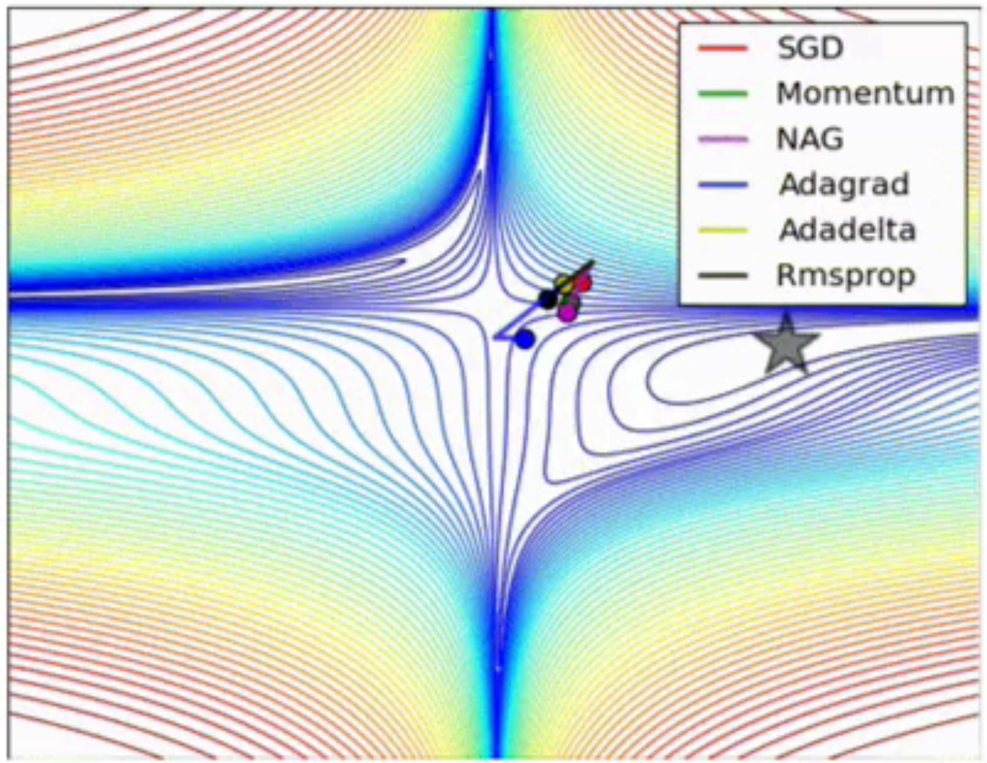
$$z_t := \beta z_{t-1} + \nabla J(W_{t-1} - \beta z_{t-1})$$



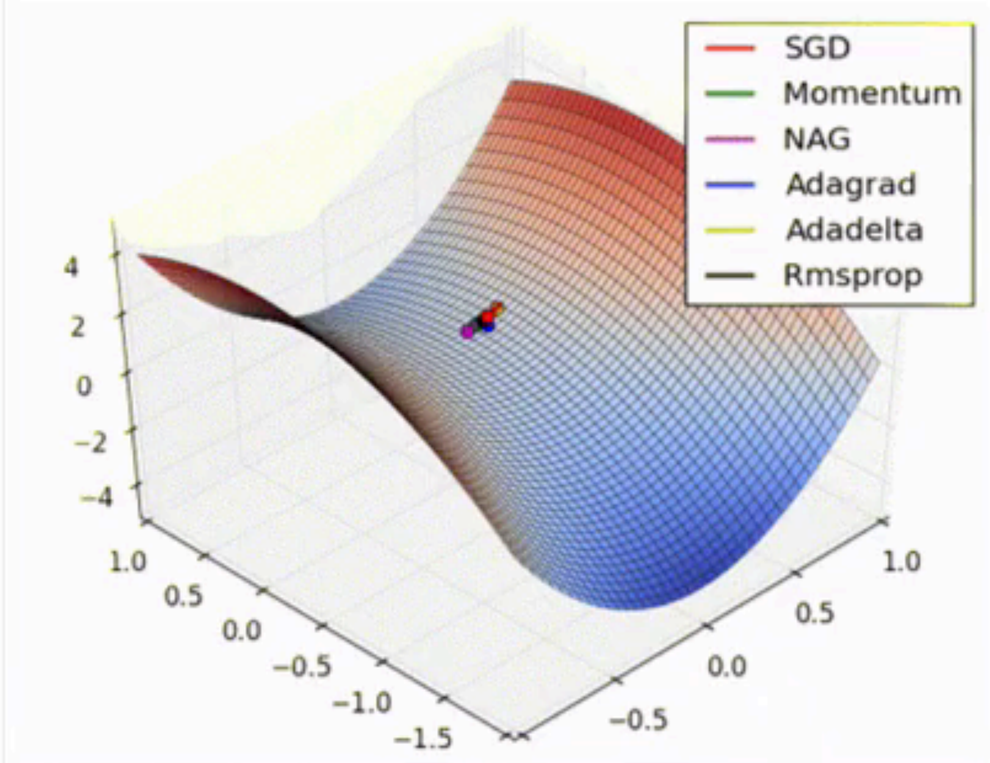
- Source: http://cs231n.github.io/neural-networks-3/

Adaptive methods

- AdaGrad
- AdaDelta
- RMSprop
- ADAM



Contour plot of gradient update methods converging on good parameters.
Figure by **Alec Radford**



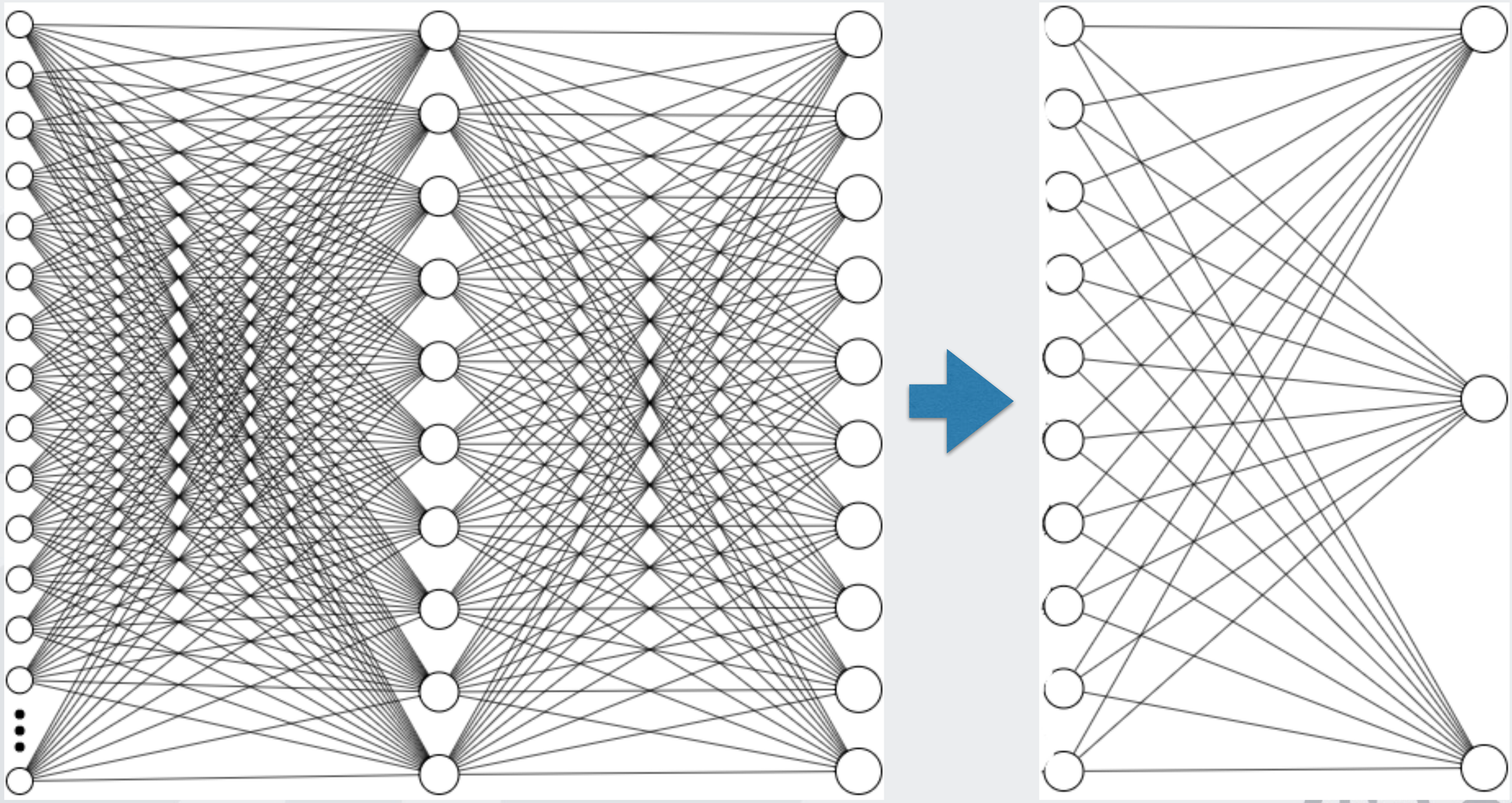
Comparison of gradient update methods escaping from a saddle point.
Notice that SGD gets stuck. Figure by **Alec Radford**

Further reading

- https://ml4a.github.io/ml4a/how_neural_networks_are_trained/
- http://ruder.io/optimizing-gradient-descent/index.html
- http://cs231n.github.io/neural-networks-3/
- http://neuralnetworksanddeeplearning.com/chap2.html

Next session

- Transfer learning and feature extraction



Assignment

- Find or compile a small labeled dataset which is interesting to you, and attempt to learn a classification or regression over it. Try to maximize accuracy on it by trying different regularization parameters and using different optimizers, as well as experimenting with different architectures, and report on the results. How high of an accuracy can you achieve?