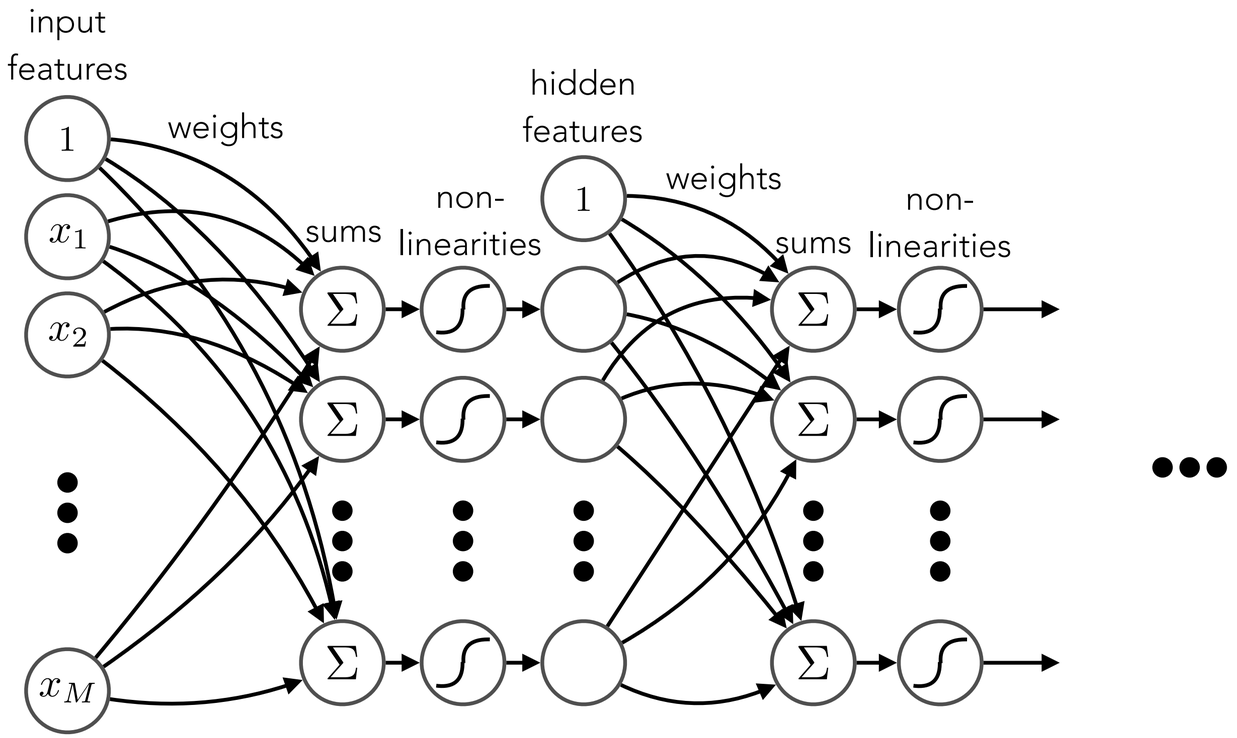
PHYS 139/239: Machine Learning in Physics

Lecture 6: Neural network optimization

Javier Duarte — January 26, 2023

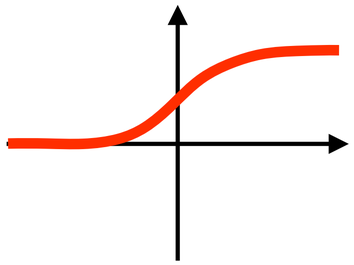
Neural networks



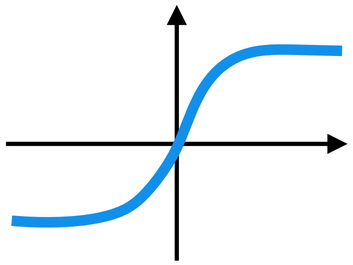
network: *sequence of parallelized weighted sums and non-linearities*

Nonlinearities

logistic sigmoid



hyperbolic tangent (tanh)

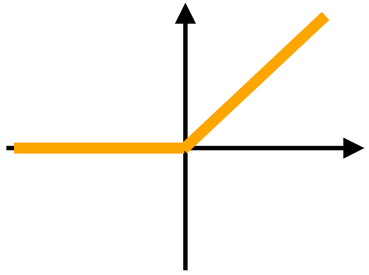


saturating

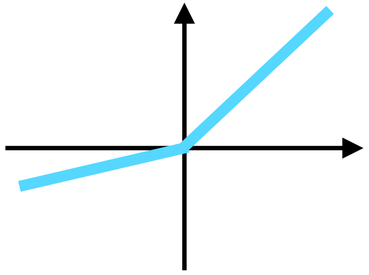
derivative goes to

<u>zero</u> at +∞ and -∞

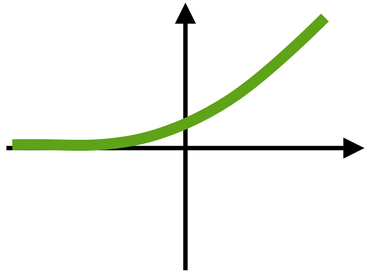
rectified linear unit (ReLU)



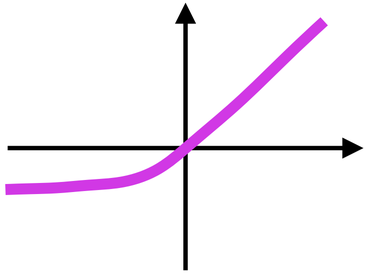
leaky ReLU



softplus



exponential linear unit (ELU)



non-saturating

<u>non-zero</u> derivative

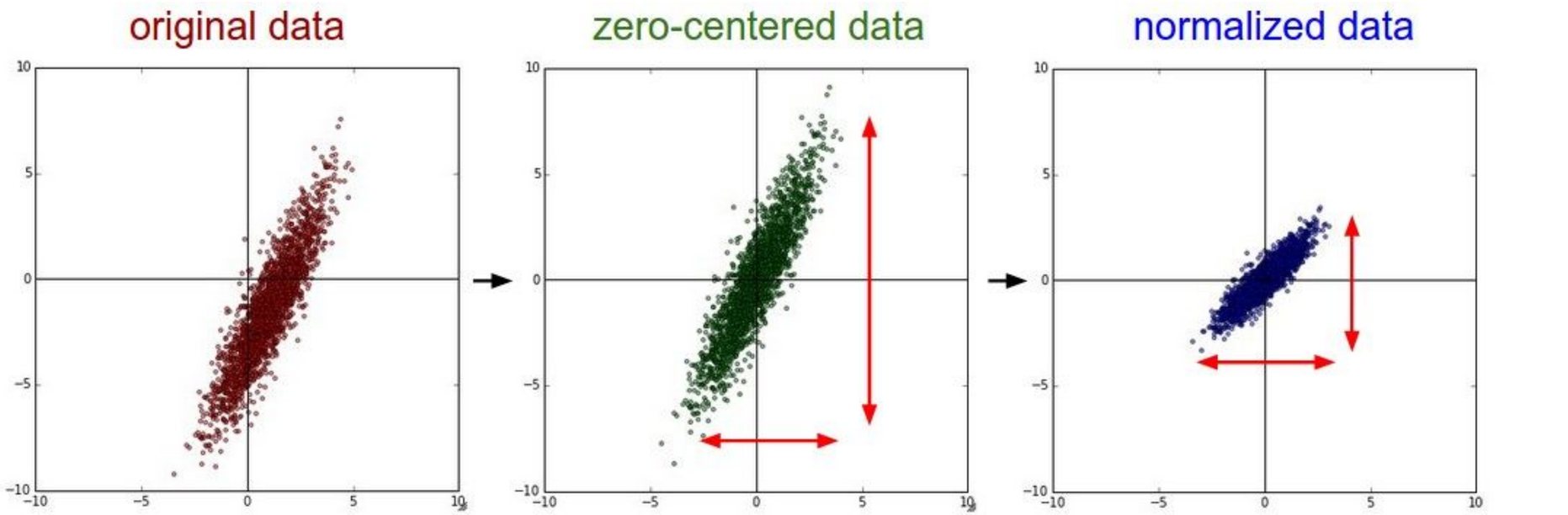
at +∞ and/or -∞

most often used

$$\mathrm{ReLU}(x) := \max(0, x)$$

Data preprocessing

- Generally good practice to preprocess data to have mean 0, and standard deviation 1 (i.e. standardized)

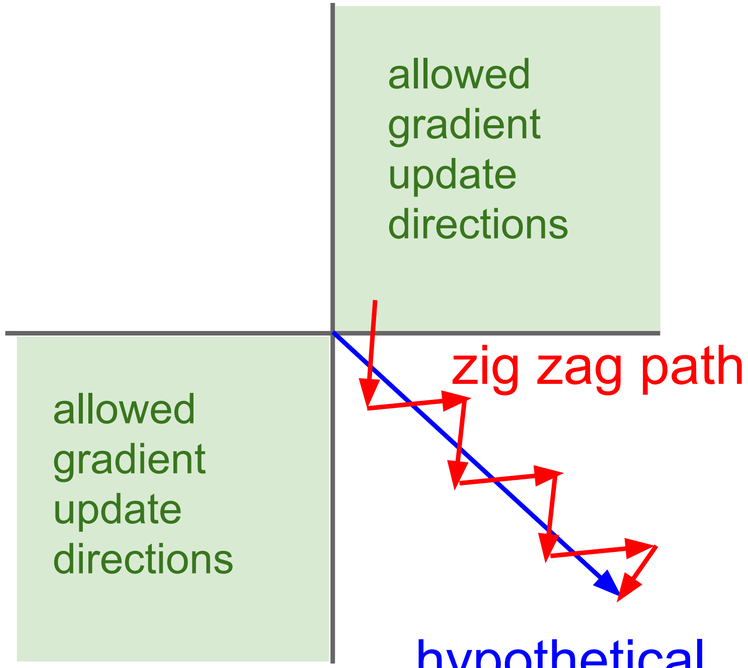


(Assume X [NxD] is data matrix,
each example in a row)

Data preprocessing: why?

Remember: Consider what happens when the input to a neuron is always positive...

$$f\left(\sum_i w_i x_i + b\right)$$



What can we say about the gradients on **w**?

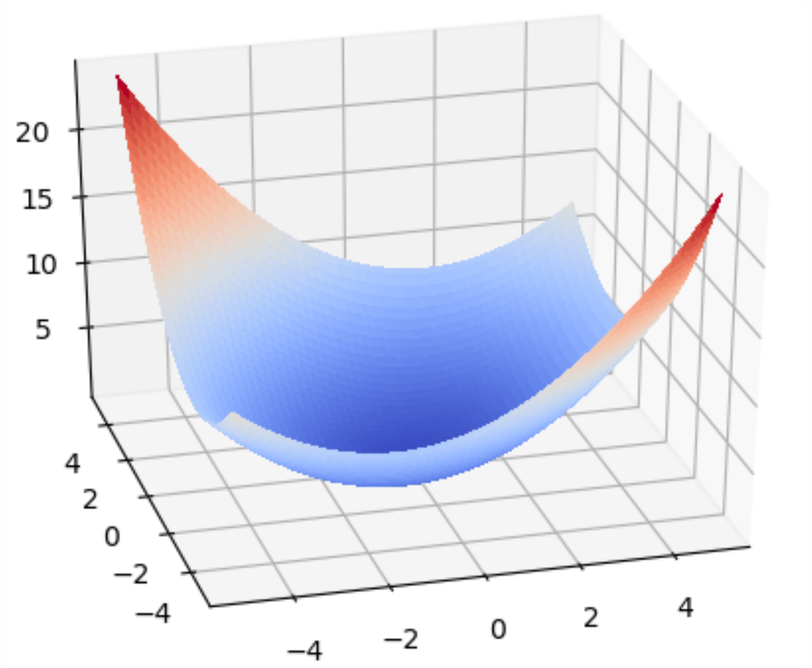
Always all positive or all negative :(

(this is also why you want zero-mean data!)

Loss landscape of NNs

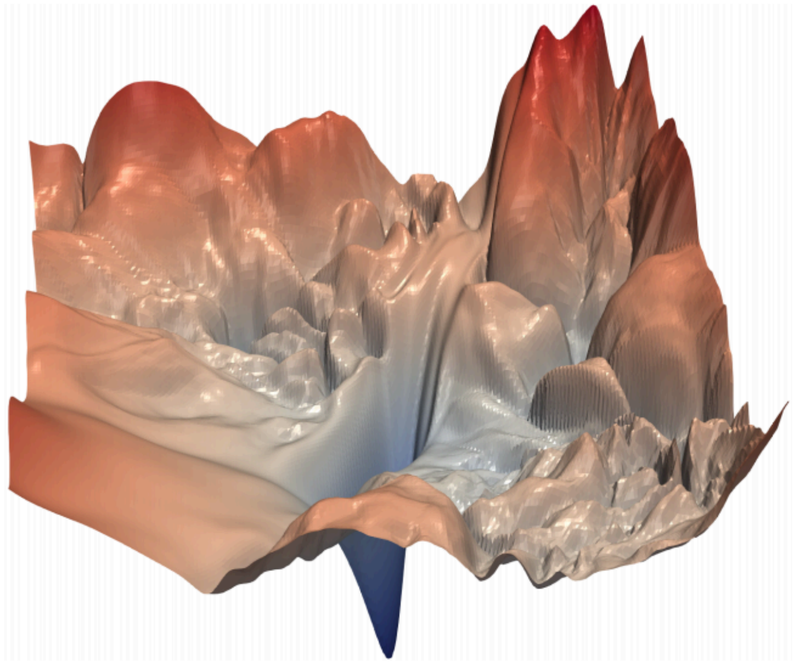
Convex problem
(logistic regression, SVMs)

$L(w)$



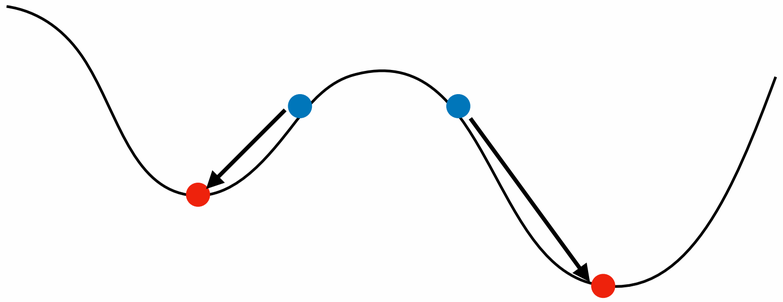
Deep Networks

$L(w)$

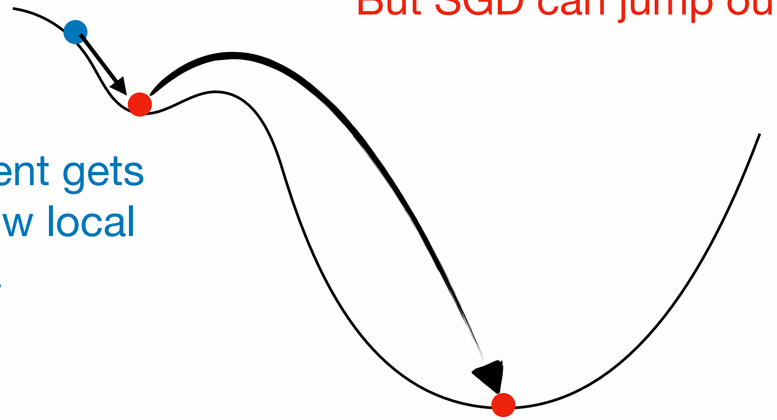


Consequences of nonconvexity

Sensitivity to initialization: based on where you start you may end up in different minima



Shallow minima: we may get stuck in a suboptimal local minimum



The noise of stochastic gradient descent is actually a benefit in deep learning!

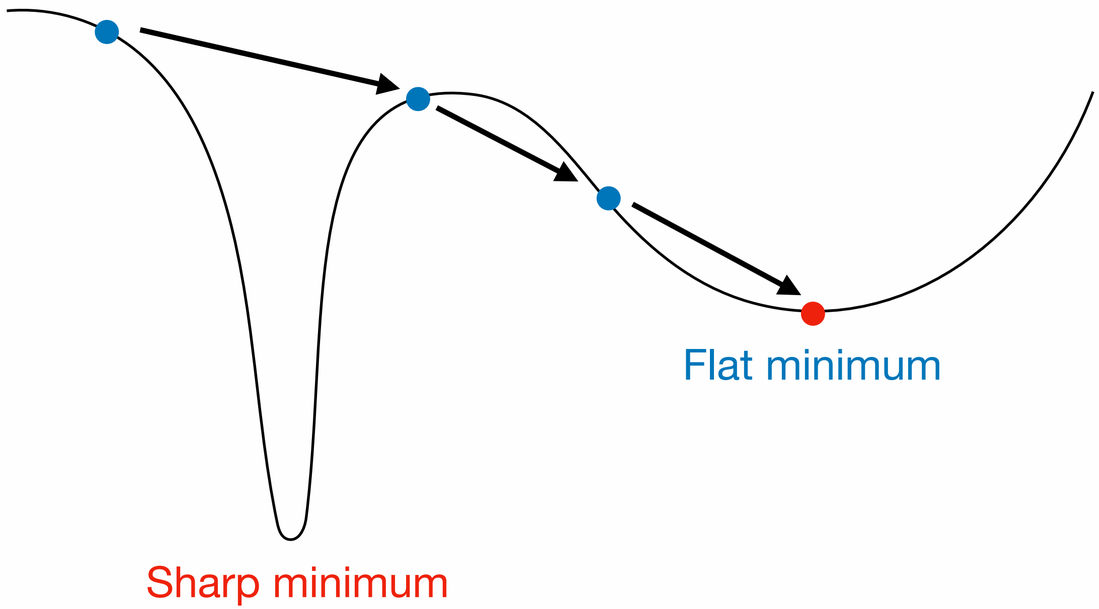
Flat & sharp minima

To converge to a minimum we need:

$$\eta < \frac{2}{\text{curvature}}$$

The noise of SGD makes us jump out of sharp minima

If the learning rate is too large we can't enter a sharp minimum

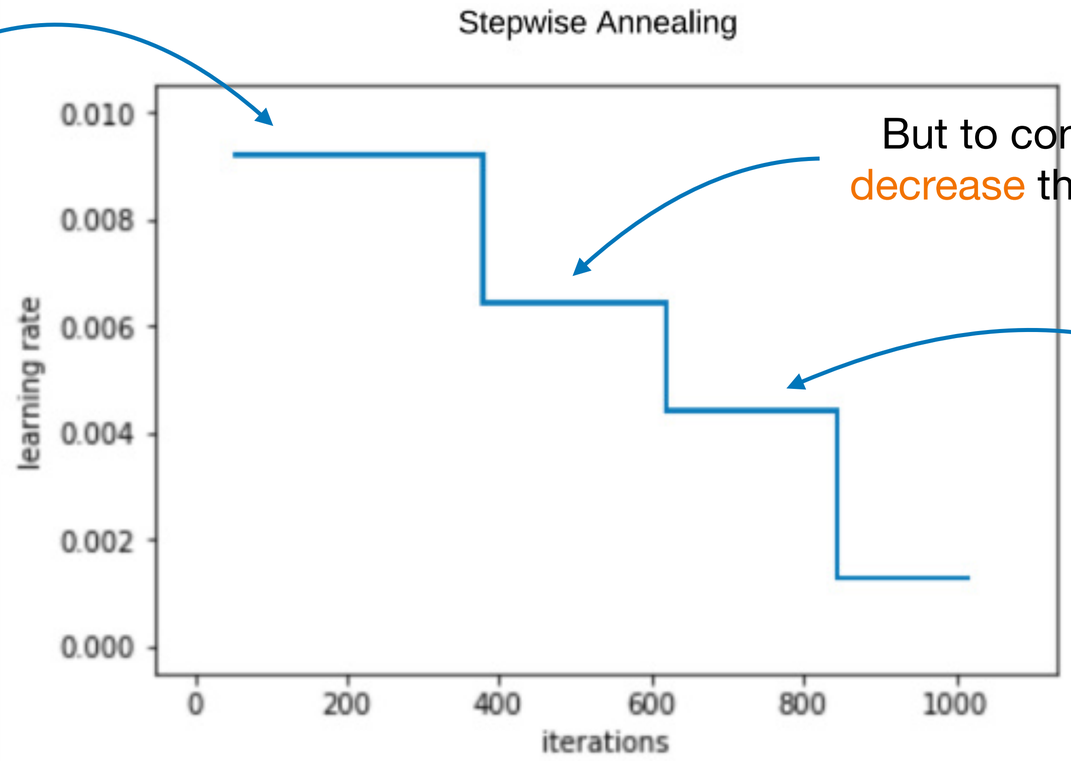


Is this a problem? In deep learning it is often observed that flat minima are better solutions, so avoiding sharp minima is good!

Learning rate annealing

We start with an high learning rate

Converges faster and avoids sharp minima



But to converge we need to decrease the learning rate later

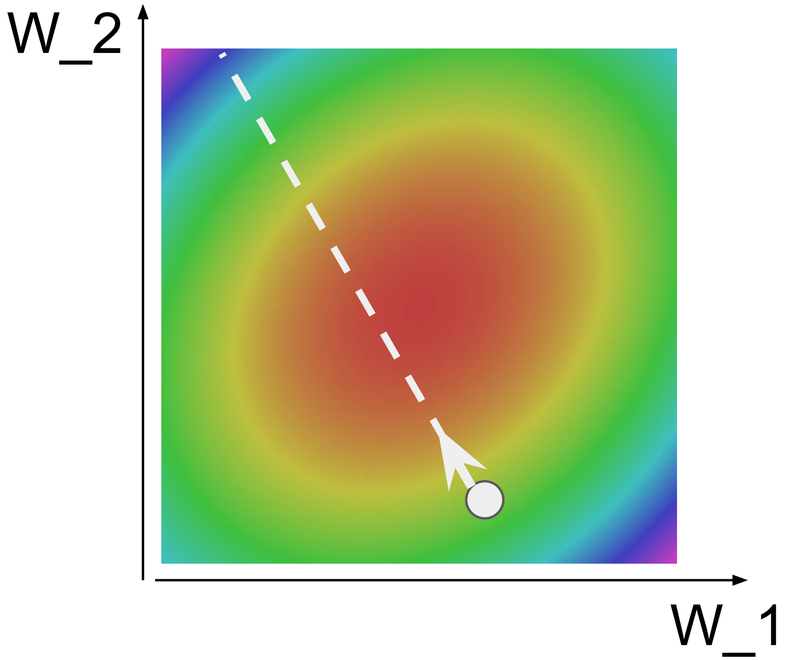
If we decrease too fast we end up in a bad minimum, so we do it in multiple steps

But are there more sophisticated optimization algorithms than SGD (with variable learning rates)?

Vanilla SGD

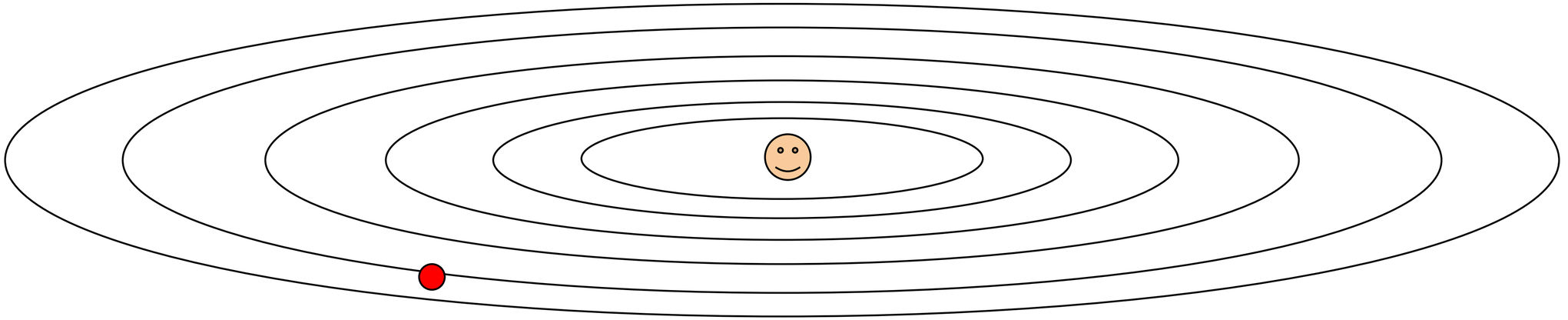
SGD:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$



Problems with vanilla SGD

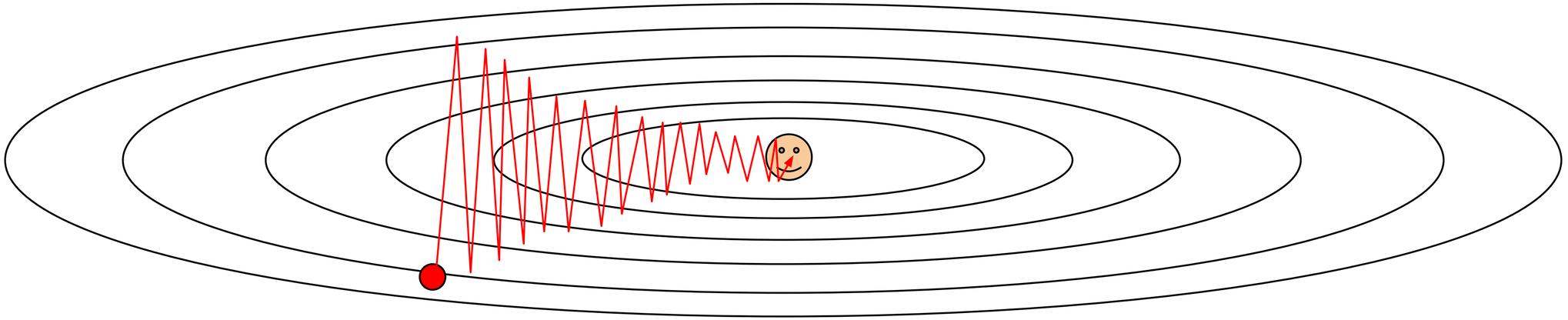
What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?



Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with vanilla SGD

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



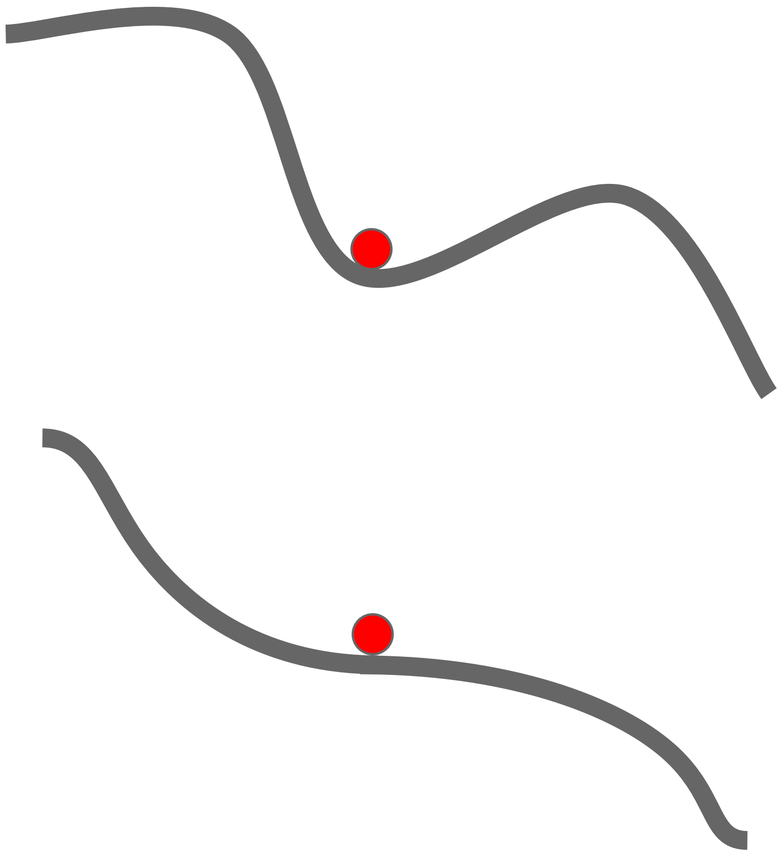
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Problems with vanilla SGD

What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck

Saddle points much more common in high dimension

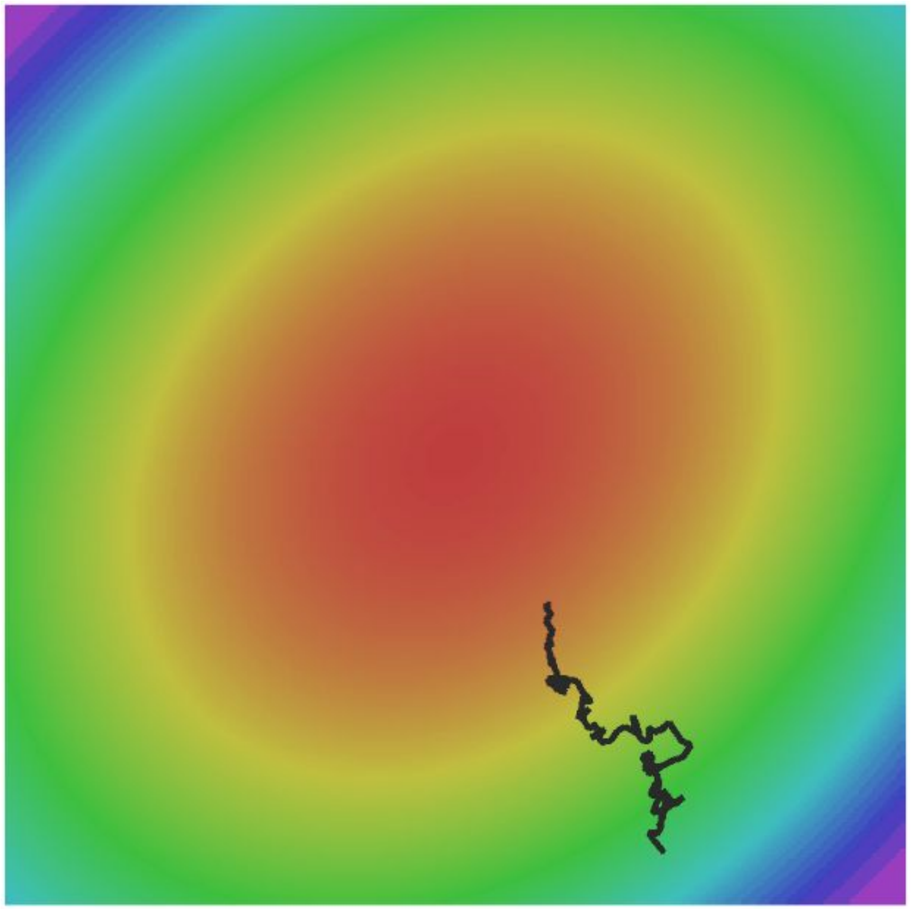


Problems with vanilla SGD

Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

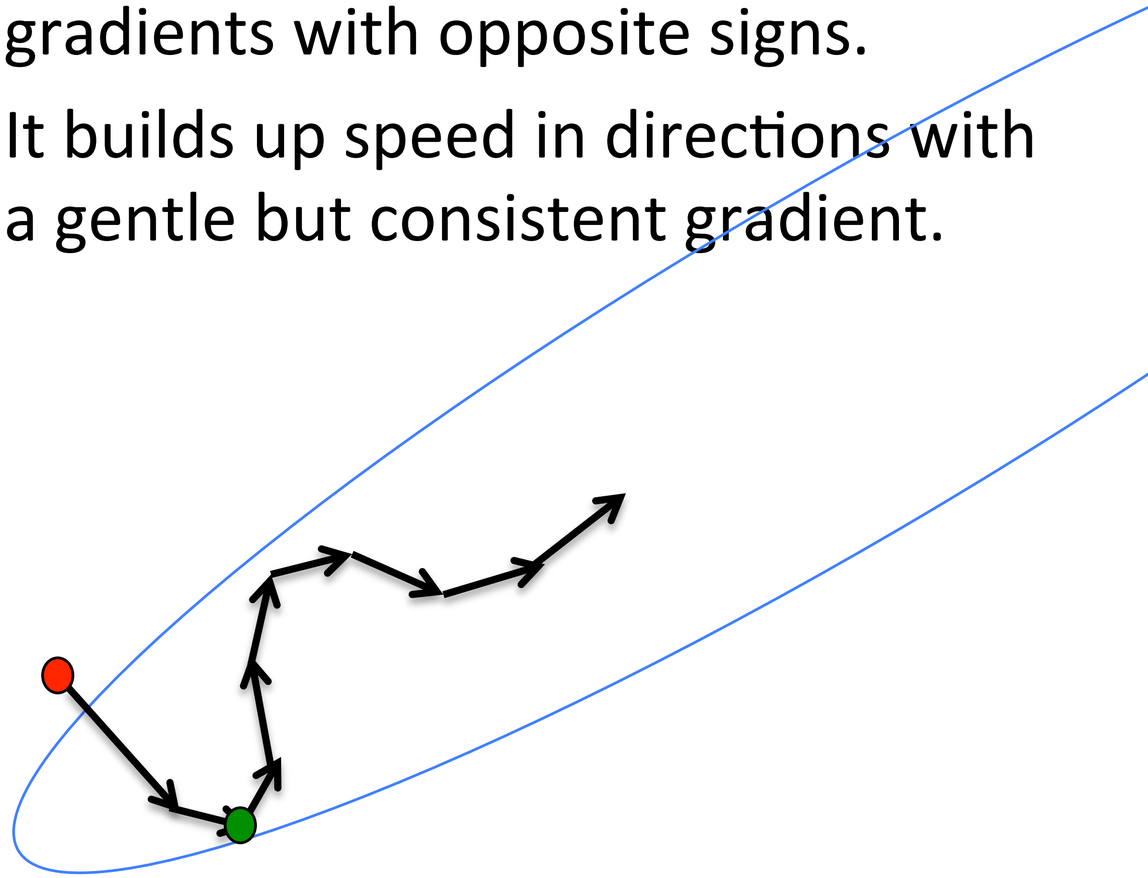


Momentum

Imagine a ball on the error surface. The location of the ball in the horizontal plane represents the weight vector.

- The ball starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
- Its momentum makes it keep going in the previous direction.

- It damps oscillations in directions of high curvature by combining gradients with opposite signs.
- It builds up speed in directions with a gentle but consistent gradient.



SGD + momentum update

SGD:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

SGD + momentum:

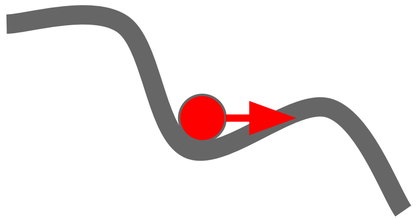
$$v_{t+1} = \rho v_t - \eta \nabla L(w_t)$$

$$w_{t+1} = w_t + v_{t+1}$$

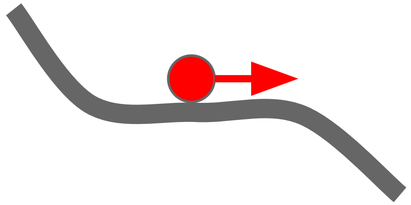
- Build up “velocity” as a running mean of gradients
- Effect of the gradient is to increment the previous velocity
- Velocity also decays by ρ, usually 0.9 or 0.99
  - Can be thought of as “friction”
- The weight change is equal to the current velocity

SGD + momentum behavior

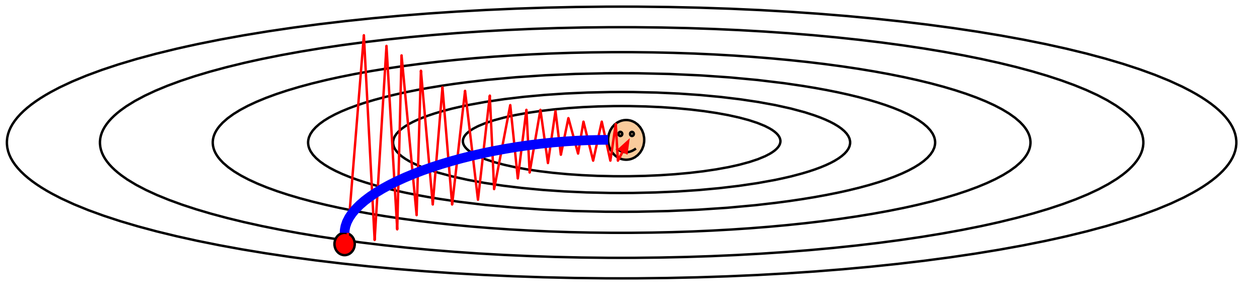
Local Minima



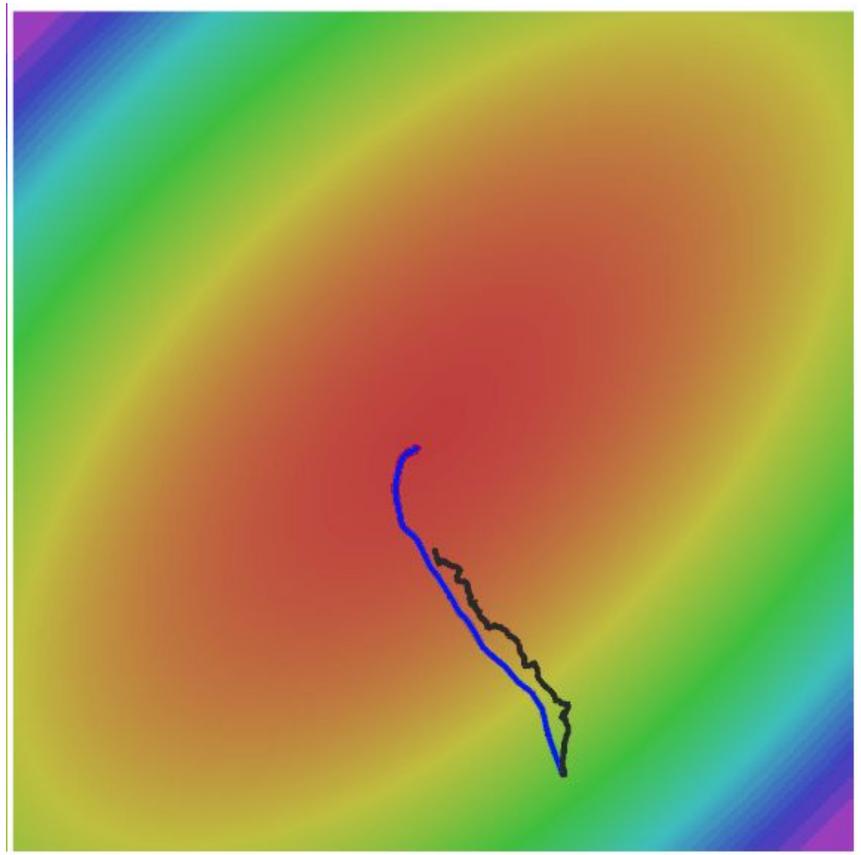
Saddle points



Poor Conditioning



Gradient Noise

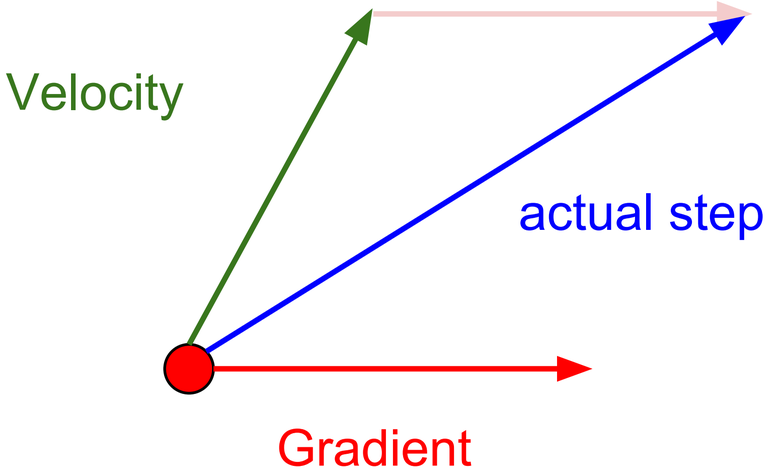


Better type of momentum: Nesterov

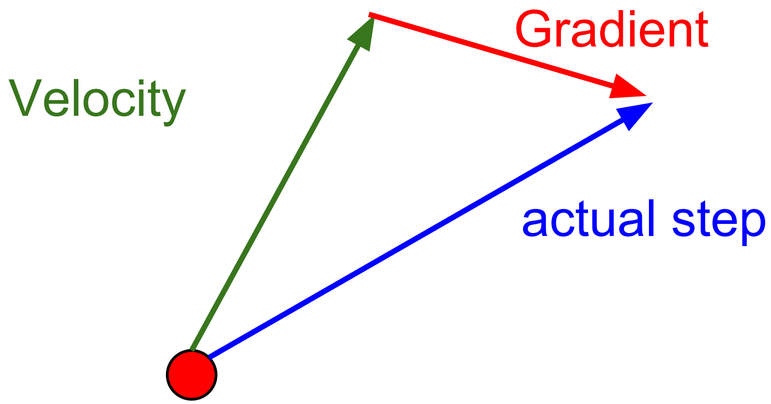
- Standard momentum method first computes the gradient at the current location and then takes a big jump in the direction of the updated accumulated gradient
- Ilya Sutskever (2012) suggested a new form of momentum that works better
  - Inspired by the Nesterov method for optimizing convex functions
- First make a big jump in the direction of the previous accumulated gradient.
- Then measure the gradient where you end up and make a correction
  - It's better to correct a mistake after you have made it!

Nesterov momentum update

Momentum update:



Nesterov Momentum



SGD + momentum:

$$v_{t+1} = \rho v_t - \eta \nabla L(w_t)$$

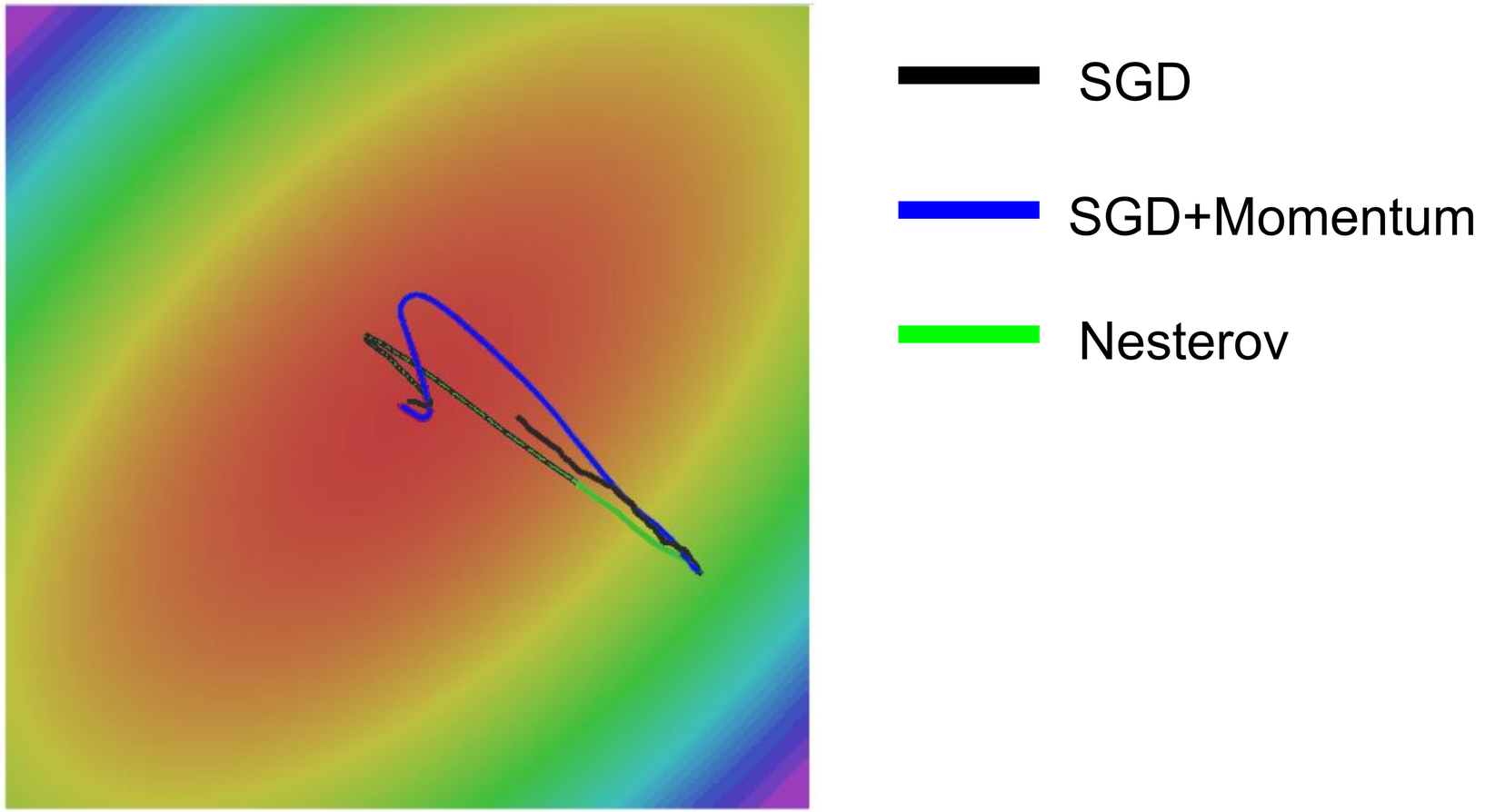
$$w_{t+1} = w_t + v_{t+1}$$

SGD + Nesterov momentum:

$$v_{t+1} = \rho v_t - \eta \nabla L(w_t + \rho v_t)$$

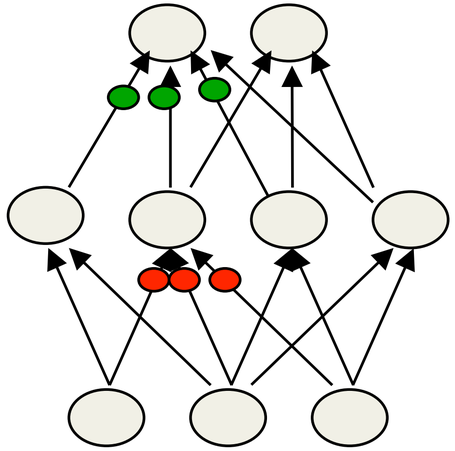
$$w_{t+1} = w_t + v_{t+1}$$

SGD + Nesterov momentum behavior



Separate, adaptive learning rates

- In a multilayer net, the appropriate learning rates can vary widely between weights:
  - The magnitudes of the gradients are often very different for different layers, especially if the initial weights are small.
  - The fan-in of a unit determines the size of the "overshoot" effects caused by simultaneously changing many of the incoming weights of a unit to correct the same error.
- So use a global learning rate (set by hand) multiplied by an appropriate local gain that is determined empirically for each weight.



Gradients can get very small in the early layers of very deep nets.

The fan-in often varies widely between layers.

AdaGrad

- AdaGrad normalizes the learning rate per parameter based on the sum of squares of previous gradients g_t
- ϵ is typically 10^{-8} to prevent division by zero

SGD update:

$$w_{t+1,i} = w_{t,i} - \eta g_{t,i}$$

$$g_{t,i} = [\nabla L(w_t)]_i$$

AdaGrad update:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

$$G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^\intercal$$

RMSProp

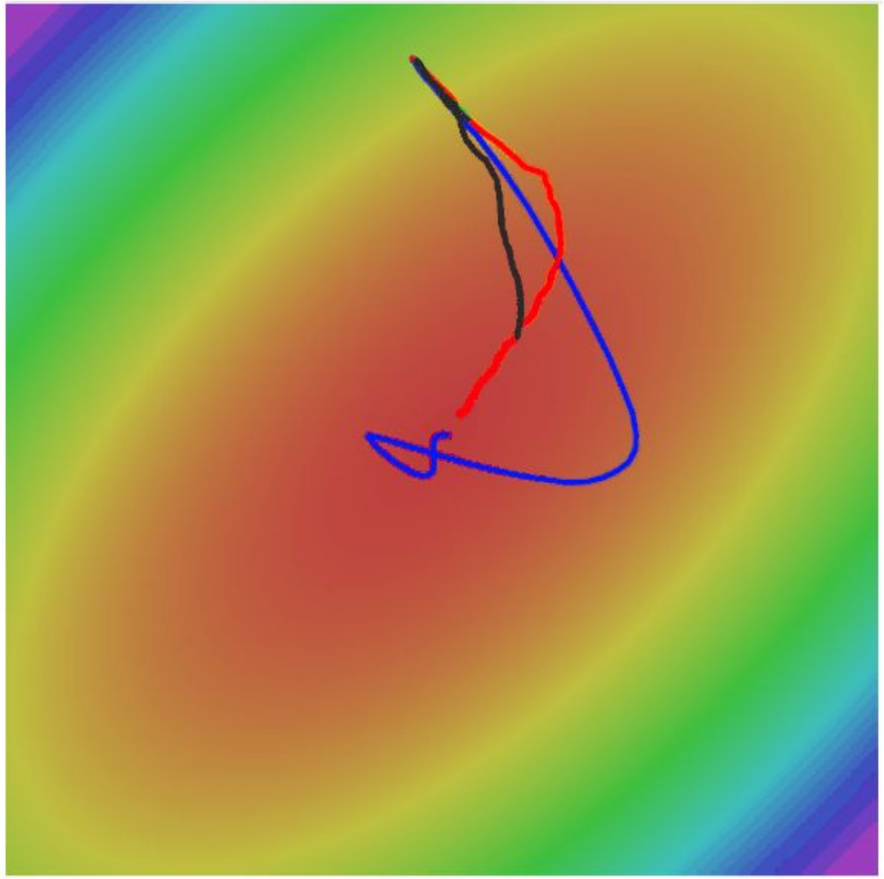
- RMSProp (and AdaDelta) is similar, but uses a running average with a decay rate of γ instead of saving all previous gradients
- γ is usually set to 0.9

RMSProp update:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

RMSProp behavior



SGD

SGD+Momentum

RMSProp

Adam

- Adaptive moment estimation (Adam) combines many of the previous tricks
- m_t and v_t estimate the 1st moment (mean) and 2nd moment (uncentered variance) of the gradients
- As m_t and v_t are initialized as 0s, they are biased towards 0, especially initially and when β_1 and β_2 are close to 1
- Counteract these biases by computing bias-corrected first and second moment estimates $\hat{m}_t$ and $\hat{v}_t$
- Typically, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

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

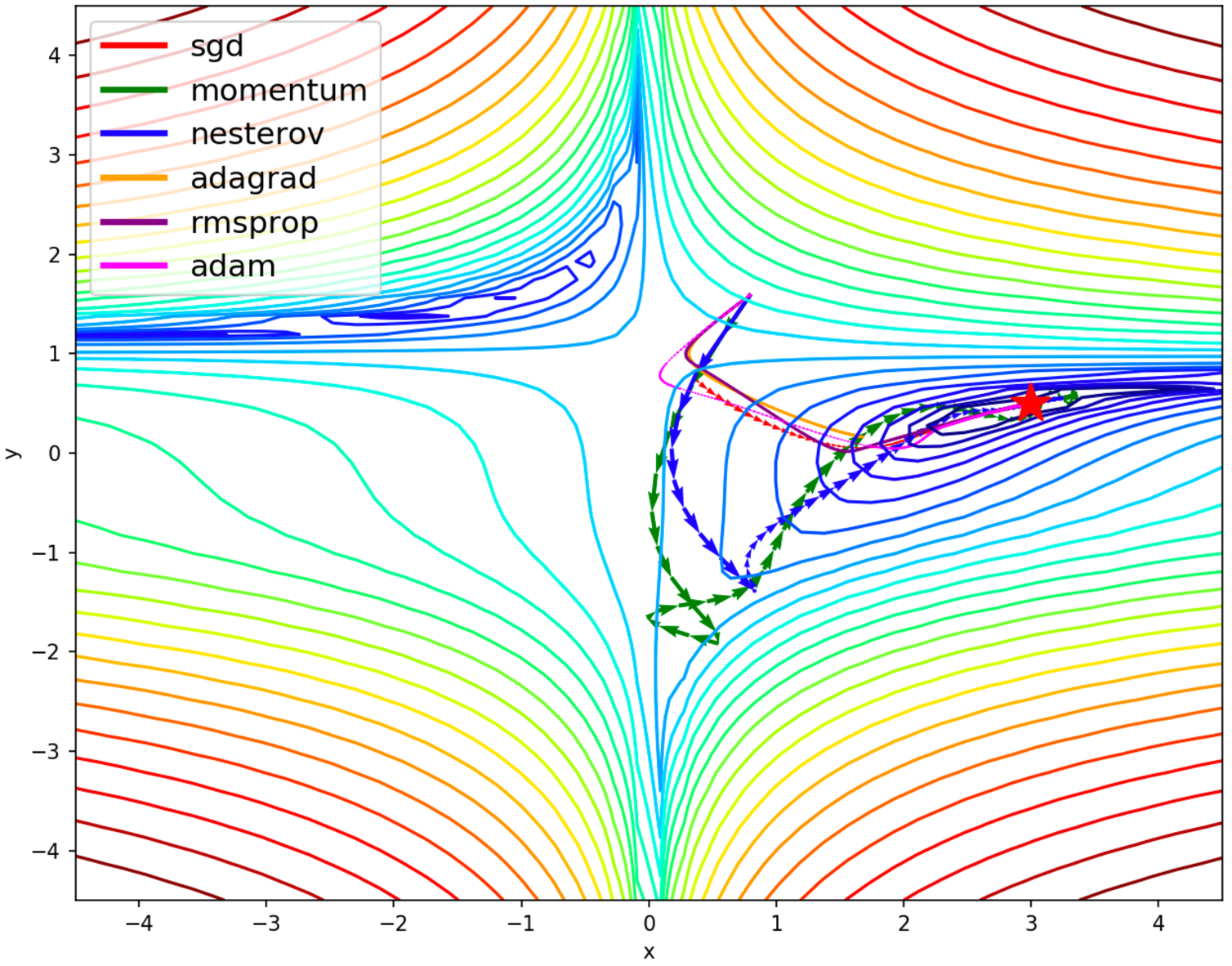
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Adam update:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

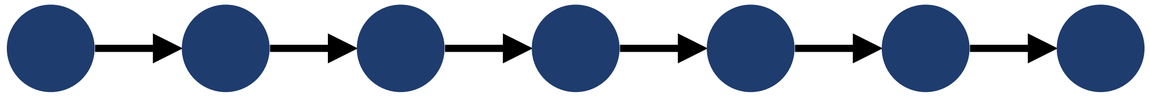
Comparison

Momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction (prefers flat minima in the error surface)



Residual connections

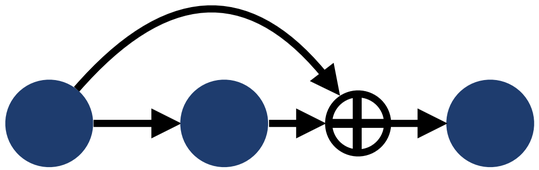
sequential connectivity: *information must flow through the entire sequence to reach the output*



information may not be able to propagate easily

→ *make shorter paths to output*

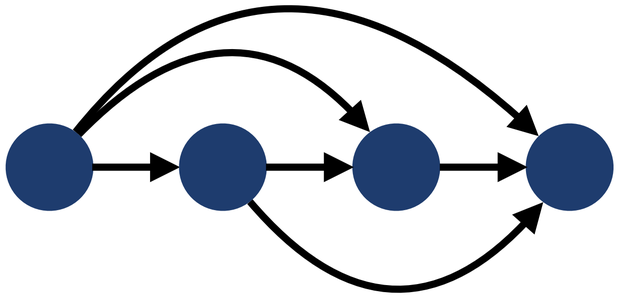
residual & highway connections



Deep residual learning for image recognition, He et al., 2016

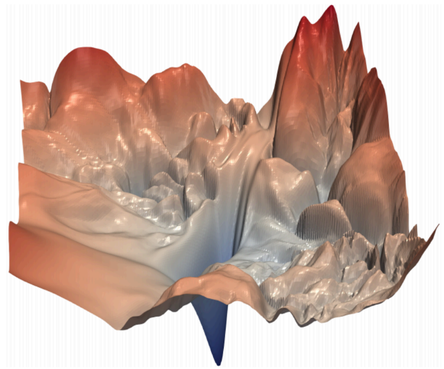
Highway networks, Srivastava et al., 2015

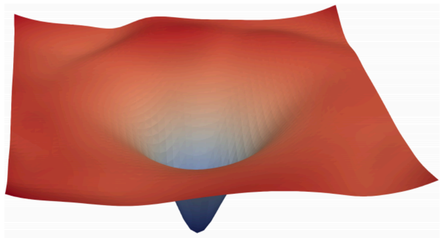
dense (concatenated) connections



Densely connected convolutional networks, Huang et al., 2017

Without residual connections





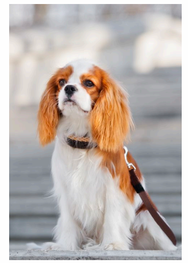
With residual connections

Data memorization

Given a training dataset with millions of completely random labels, DNNs networks can easily reach zero training error.

→ Monkey

→ Salamander



→ Wine bottle

They do so my memorizing the association between meaningless but unique patterns in the samples and the label.

if the image contains this patch:  then output: Monkey

The problem is that they learn these degenerate patterns even on real data…
(which is also a privacy risk)

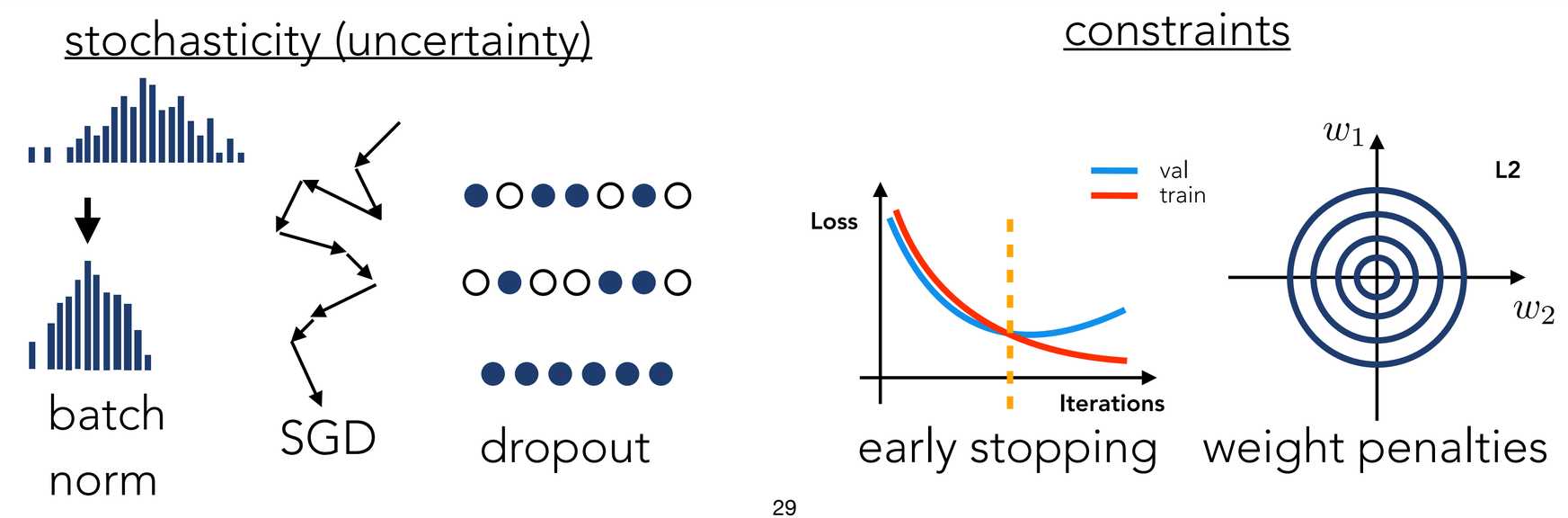
Generalization bounds

One can show that the "generalization gap" is bounded by the amount of information memorized by the network:

$$L_{\text{test}} - L_{\text{train}} \leq \sqrt{\frac{I(w;D)}{N}}$$

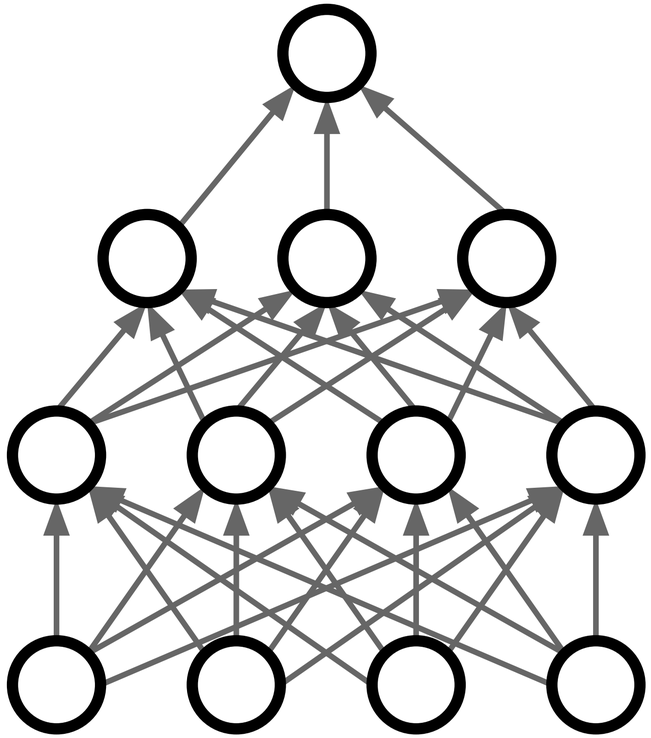
Information that the weights contain about the training examples

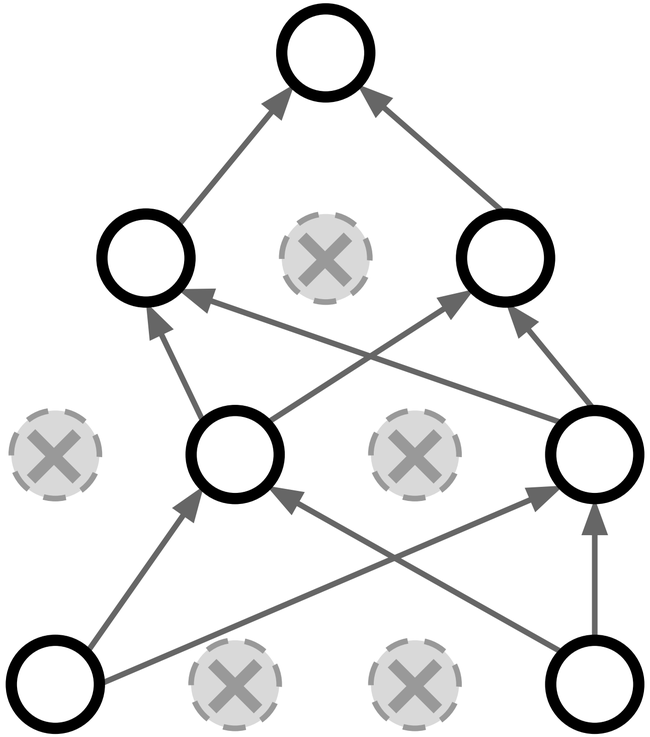
Ways to limit the information stored in the weights:



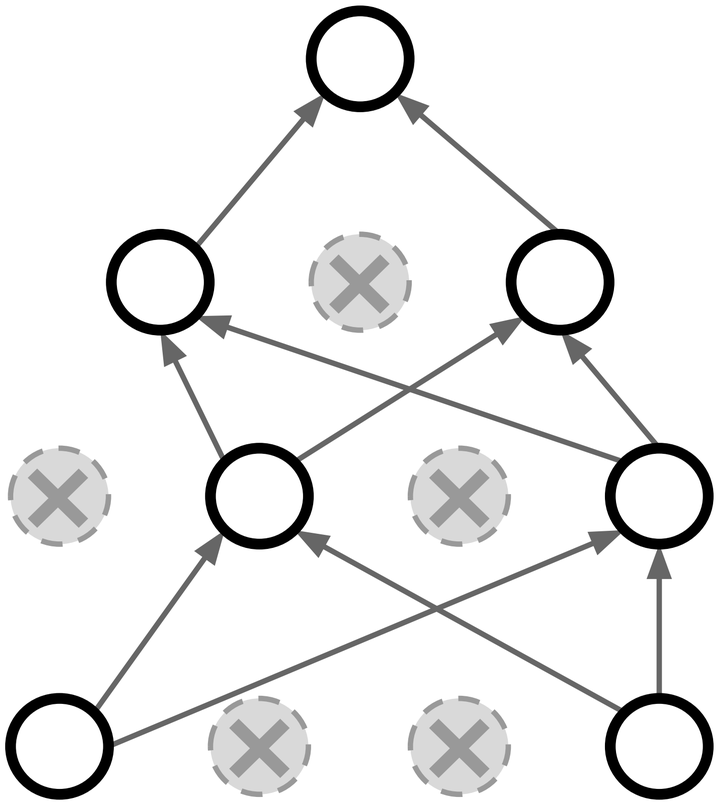
Dropout

- During training, randomly set some neurons to zero
- Probability of dropping is a hyper parameter: 0.5 is common

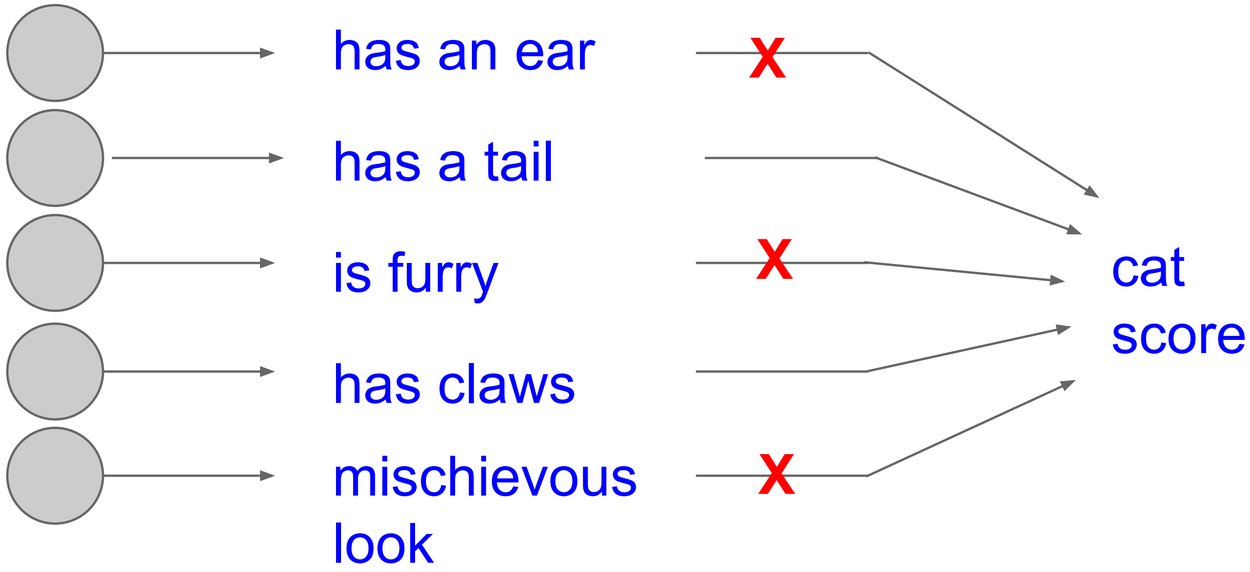




Dropout



Forces the network to have a redundant representation;
Prevents co-adaptation of features



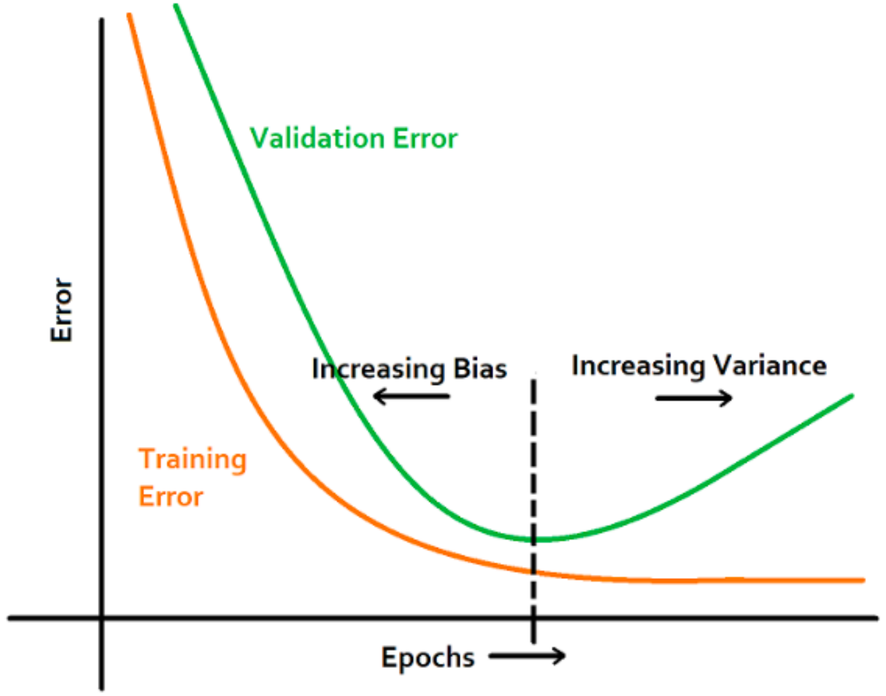
Dropout is training a large **ensemble** of models (that share parameters).

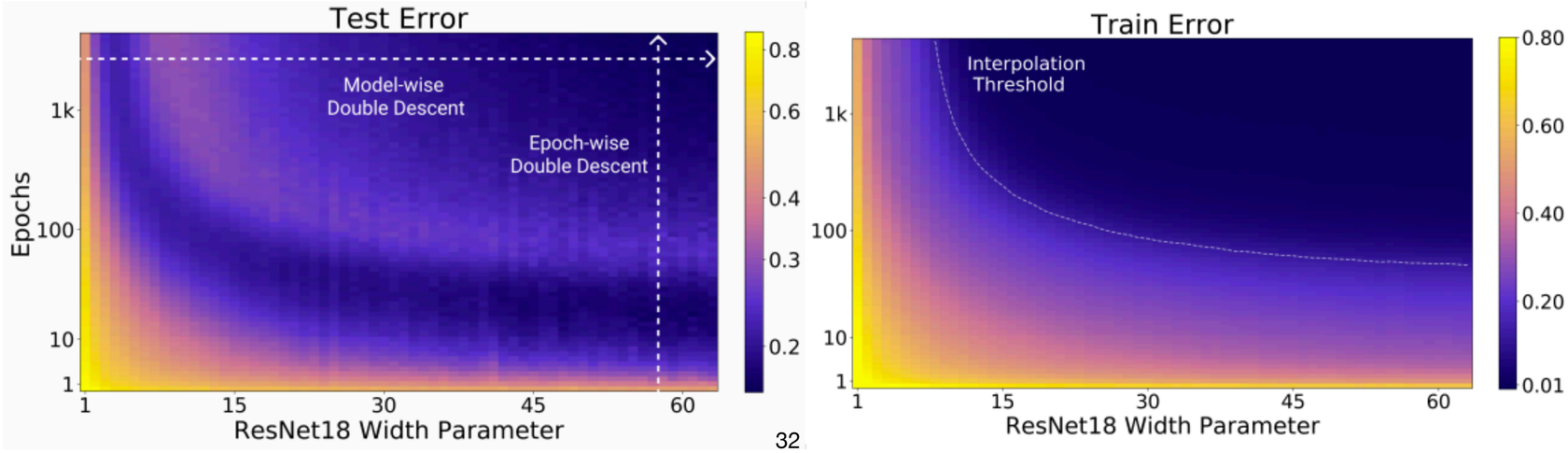
Each binary mask is one model

An FC layer with 4096 units has
$2^{4096} \sim 10^{1233}$ possible masks!
Only $\sim 10^{82}$ atoms in the universe...

Early stopping

- Monitor some important metric on a validation dataset, such as loss value or accuracy (typically compute at the end of each epoch)
- Terminate training if metric has not improved for n epochs (hyperparameter known as patience)
- But beware: “deep double descent”: openai.com/blog/deep-double-descent





Summary

- Optimization of nonconvex functions can be tricky
  - Optimizers: (Nesterov) Momentum, RMSProp, Adam, SGD with learning rate schedule
  - Model/data choices: data preprocessing, batch normalization, residual connections
  - In practice, Adam with default settings does a great job
  - SGD with optimized learning rate schedule can likely achieve “best” results
- Memorization/generalization can be an issue; regularization to the rescue!
  - Constraints: L1/L2, early stopping
  - Stochasticity: small batch size, batch normalization, dropout, data augmentations

Next time

- Structured image-like data and convolutional neural networks