Deep Learning
Episode 0

ML recap. Adaptive optimization

Maxim Borisyak, Alexander Panin, Andrey Ustyuzhanin

Yandex Data Factory

British Hedgehog Preservation Society

Linear Regression

Model:

$$X \longrightarrow Wx + b \longrightarrow Y^{pred}$$

Objective function:

$$L = \sum_i (y_i - y_i^{pred})^2$$

Optimization (exact):

$$w = (X^T X)^{-1} X^T y$$

Linear Regression

Model:

$$X \longrightarrow Wx + b \longrightarrow Y^{pred}$$

Objective function:

$$L = \sum_i (y_i - y_i^{pred})^2$$

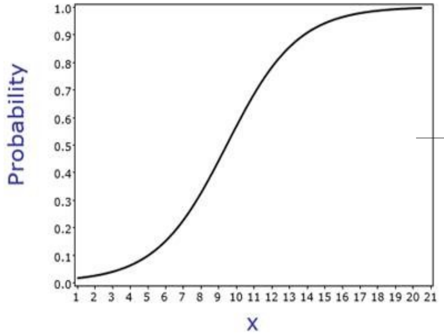
Optimization (iterative):

$$w_0 \leftarrow 0$$

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial W}$$

$$\frac{\partial L}{\partial W} = \sum_i -2x(y_i - (wx_i + b))$$

Logistic Regression

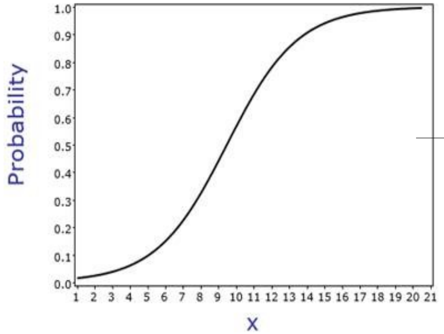
X ⟶ Wx +b ⟶  ⟶ P(y)

$$P(y) = \sigma(Wx + b)$$

Objective function ?

Logistic Regression

Model:

X → Wx +b →  → P(y|x)

Objective function:

$$L = -\sum_i y_i \log P(y|x_i) + (1 - y_i)\log(1 - P(y|x_i))$$

Optimization (iterative):

You guessed it!

Logistic Regression

Model:

$$X \longrightarrow \begin{array}{l} a_{[y=a]}=W_a x+b_a \\ a_{[y=b]}=W_b x+b_b \\ a_{[y=c]}=W_c x+b_c \end{array} \longrightarrow \frac{e^{a_{[y=class]}}}{\sum_j e^{a_{[y=j]}}} \longrightarrow \begin{array}{l} \text{P(y=a|X)} \\ \text{P(y=b|X)} \\ \text{P(y=c|X)} \end{array}$$

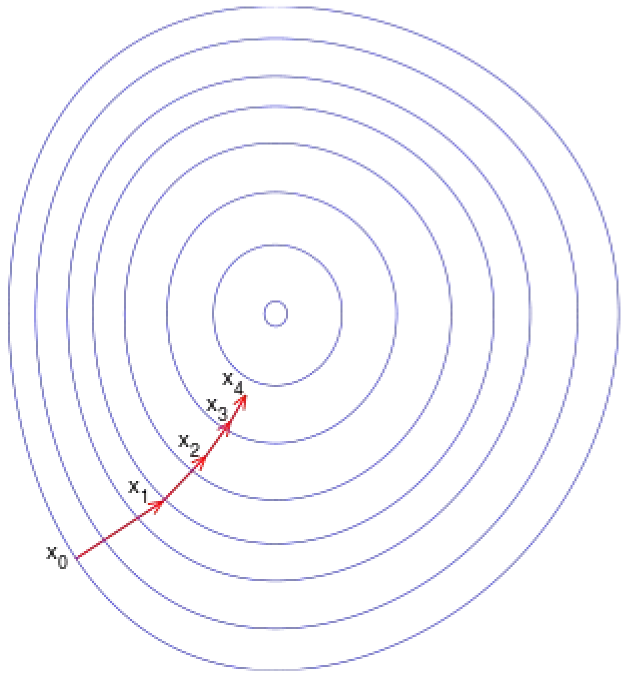
Objective function:

$$L=-\sum_i \sum_{class} [y_i=class] \log P\left(y=class|x_i\right)$$

Gradient descent

Update:

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial w}$$



- a – learning rate a<<1
- L – loss function

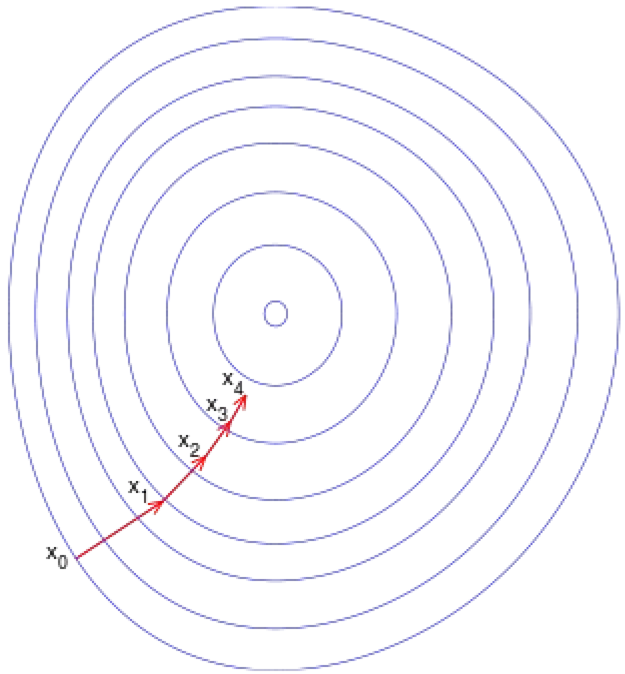
Can we do better?

Gradient descent

Update:

$$w_{i+1} \leftarrow w_i - \alpha \frac{\partial L}{\partial w}$$

- a – learning rate a<<1
- L – loss function



Can we do better?

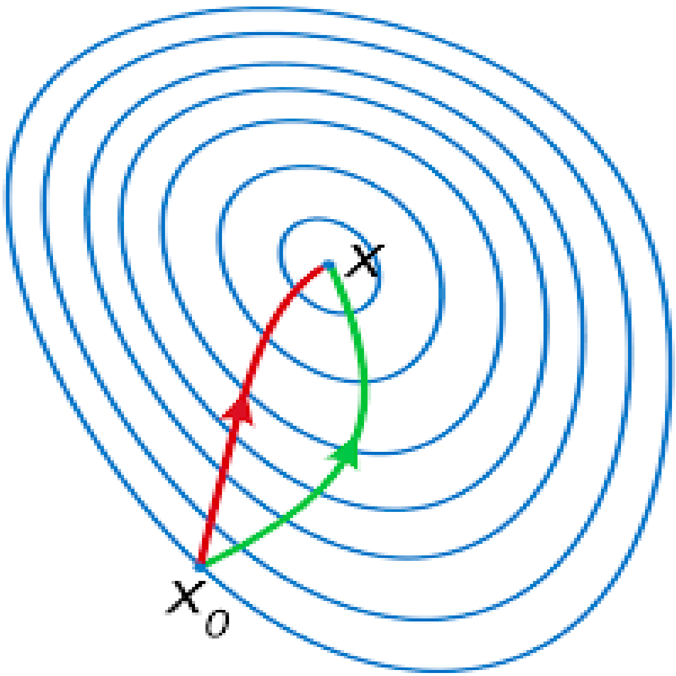
Newton-Raphson

Parameter update

$$w_{i+1} \leftarrow w_i - \alpha H_L^{-1} \frac{\partial L}{\partial w}$$

Hessian:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$



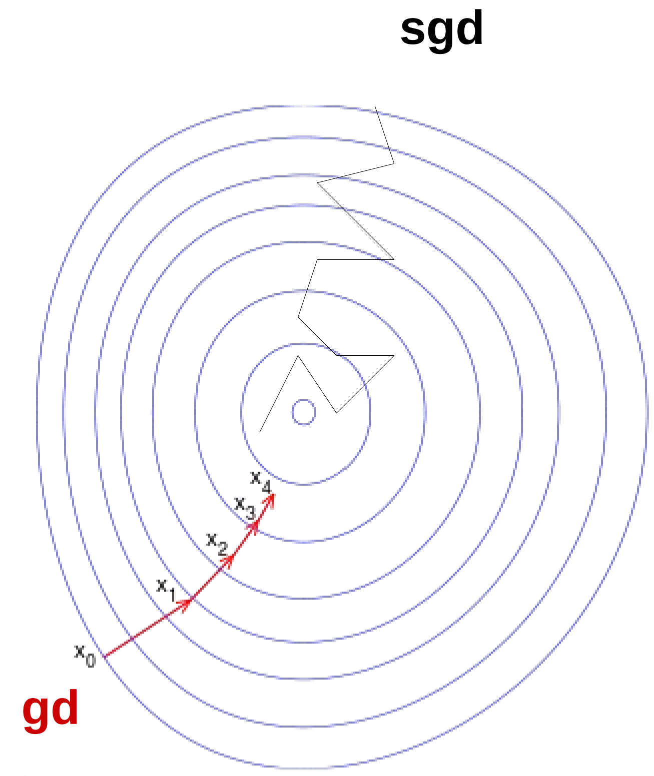
Red: Newton-Raphson
Green: gradient descent

Any drawbacks?

Stochastic gradient descent

Loss function is mean over all data samples.

Approximate with 1 or few random samples.



Update:

$$w_{i+1} \leftarrow w_i - \alpha E \frac{\partial L}{\partial w}$$

- E – expectation
- Learning rate should decrease

SGD with momentum

Idea: move towards "overall gradient direction",
Not just current gradient.

$$w_0 \leftarrow 0 ; v_0 \leftarrow 0$$

$$\nu_{i+1} \leftarrow \alpha \frac{\partial L}{\partial w} + \mu \nu_i$$

$$w_{i+1} \leftarrow w_i - \nu_{i+1}$$

Helps for noisy gradient / canyon problem

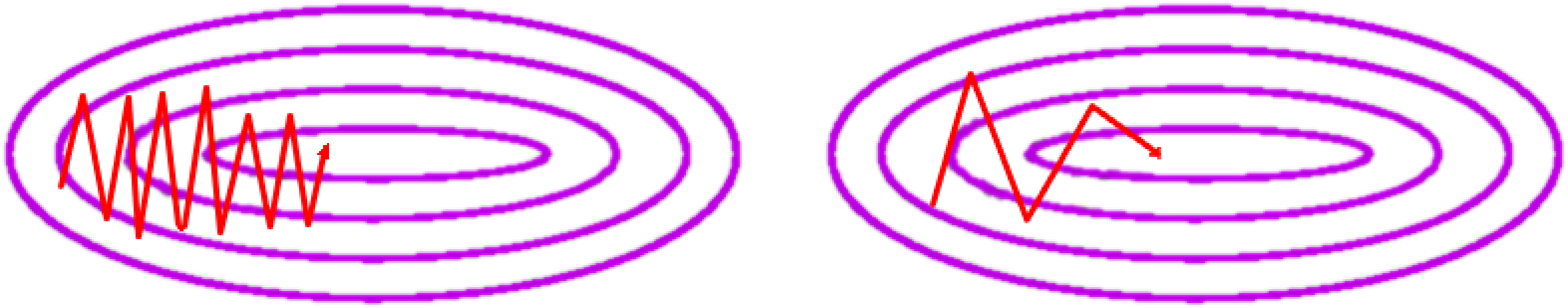
SGD with momentum

Idea: move towards “overall gradient direction”,
Not just current gradient.

$$w_0 \leftarrow 0 ; v_0 \leftarrow 0$$

$$\nu_{i+1} \leftarrow \alpha \frac{\partial L}{\partial w} + \mu \nu_i$$

$$w_{i+1} \leftarrow w_i - \nu_{i+1}$$



AdaGrad

Idea: decrease learning rate individually for each parameter in proportion to sum of it's gradients so far.

$$G_t = \sum_{\tau=1}^{t} \left[\frac{\partial L}{\partial w} \right]^2$$

"Total update path length"
(for each parameter)

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \frac{\partial L}{\partial w}$$

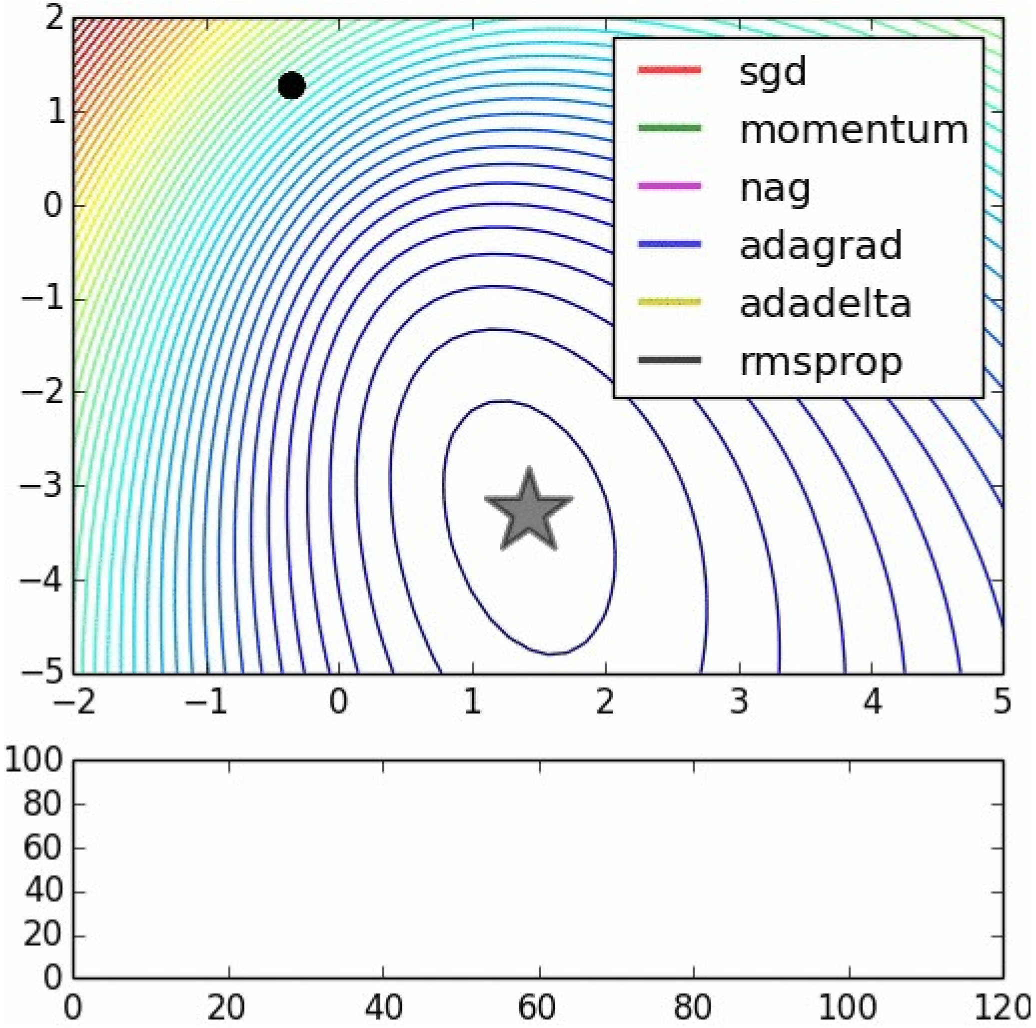
RMSProp

Idea: make sure all gradient steps have approximately same magnitude (by keeping moving average of magnitude)

$$ms_{t+1} = \gamma \cdot ms_t + (1-\gamma) \left\| \frac{\partial L}{\partial w} \right\|^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{ms+\epsilon}} \frac{\partial L}{\partial w}$$

Alltogether



Moar stuff

Without Hessian

- Adadelta ~ adagrad with window
- Adam ~ rmsprop + momentum
- Nesterov-momentum
- Hessian-free (narrow)
- Conjugate gradients

Estimate inverse Hessian

- BFGS
- L-BFGS
- ****-BFGS

Regularization (weight)

General idea:

$$L_{new} = L + reg$$

performance = how_i_fit_data + how_reasonable_i_am

L2 regularizer

$$L_{new} = L + \beta \|\theta\|_2 = L + \beta \sum_i \theta_i^2$$

linear models: theta = {w,b}

- a.k.a. weight decay
- a.k.a. Tikhonov regularizer
- a.k.a. normal prior on params

Regularization (weight)

L2 regularizer

$$L_{new} = L + \beta \sum_i \theta_i^2$$

L1 regularizer

$$L_{new} = L + \beta \sum_i |\theta_i|$$

Difference between L1, L2?

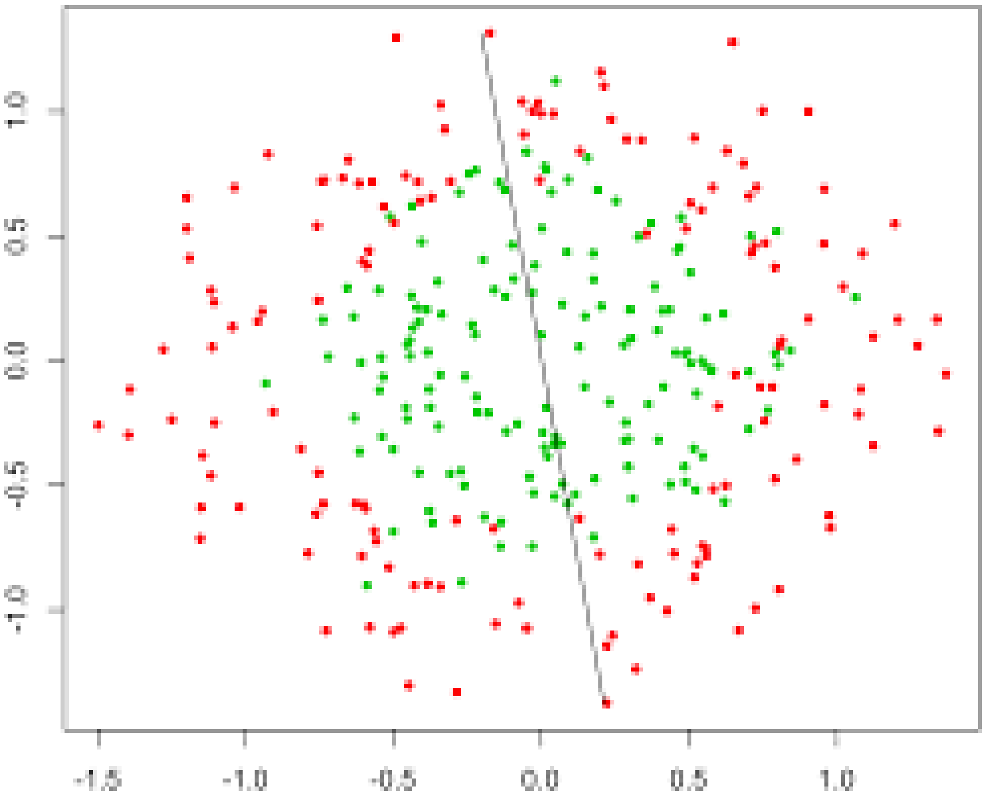
Any other way to regularize?

Regularization(other)

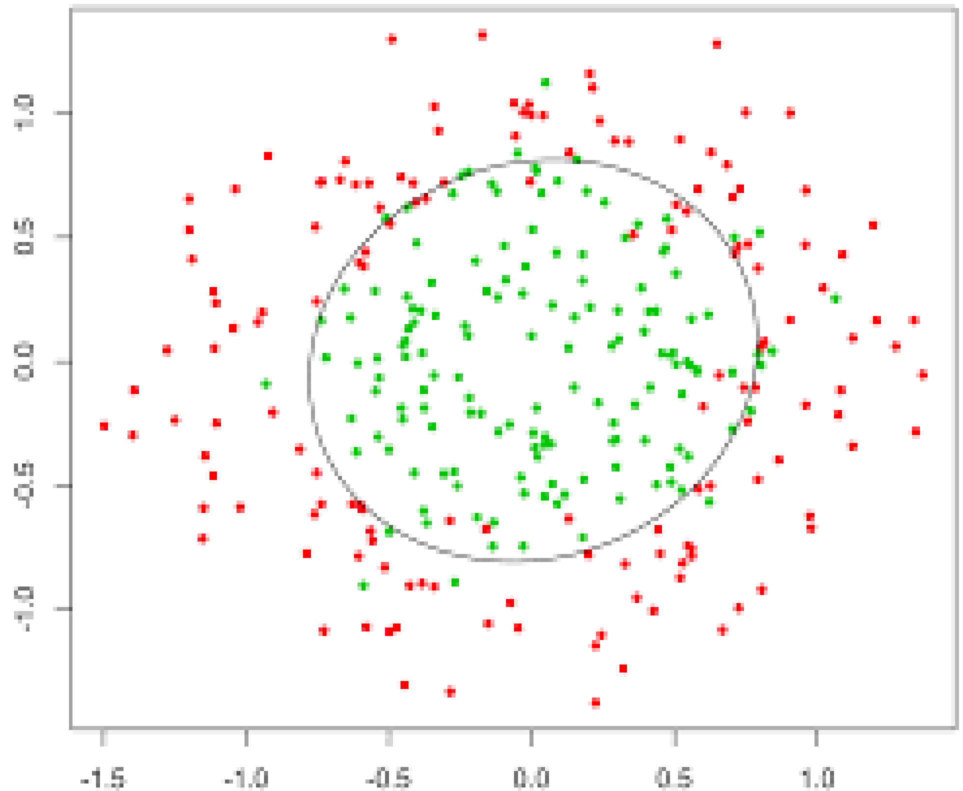
- Distort input
- Distort weights
- Additional objective
- Domain-specific stuff
- Moar data :)
- etc.

Most are domain- or model-specific

Nonlinear dependencies



What we have



What we wan

- How to get that?

Nuff

Let's go implement that!