The main difference b/t deep and shallow learning

CEMS
Chemical Engineering
& Materials Science

shallow ML → we know the features ($\vec{x}$)
(traditional)
↳ tabular data (low data regime)

Deep Learning (DL) → we learn representation using very simple features
(unstructured data)

Similar rules apply
↳ validation
↳ define success
↳ collecting good data

We will only scratch the deep learning surface

CEMS
Chemical Engineering
& Materials Science

CS231n: Deep Learning for Computer Vision

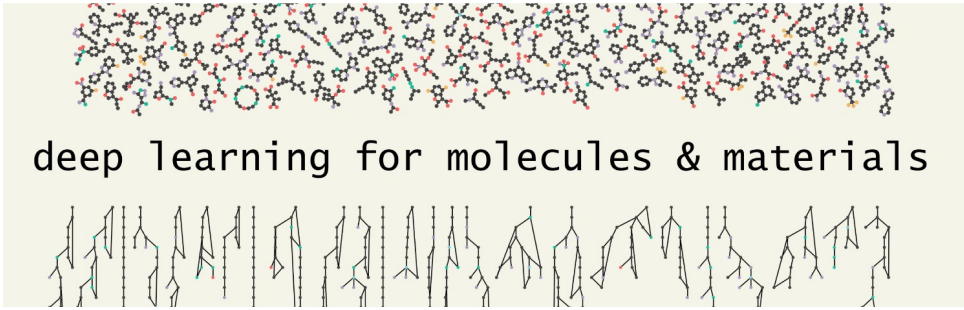
Stanford - Spring 2022

http://cs231n.stanford.edu/
https://www.youtube.com/watch?v=NfnWJUyUJYU

https://atcold.github.io/NYU-DLFL22/

https://www.youtube.com/watch?v=00s9ireCnCw

https://dmol.pub/
Andrew White



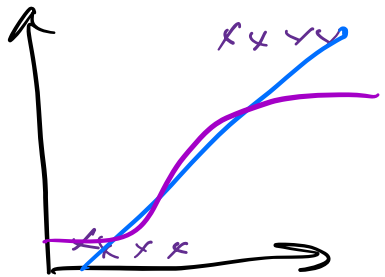
But first back to linear models

Linear models → classification → logistic regression

1) Start w/ linear model, $z = \sum w_i x_i$

2) Pass through sigmoid function

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$$

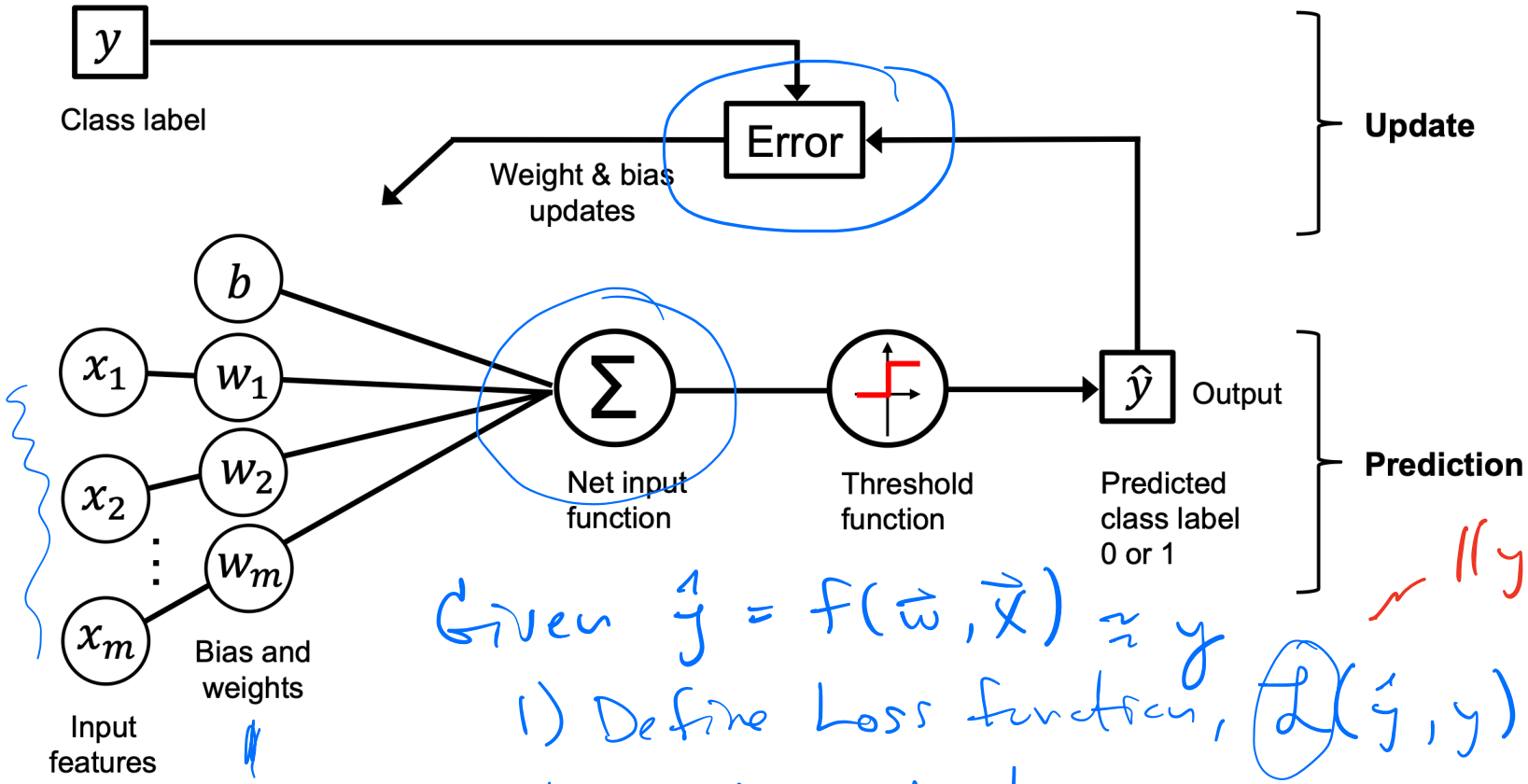


Generalized linear models (GLM)

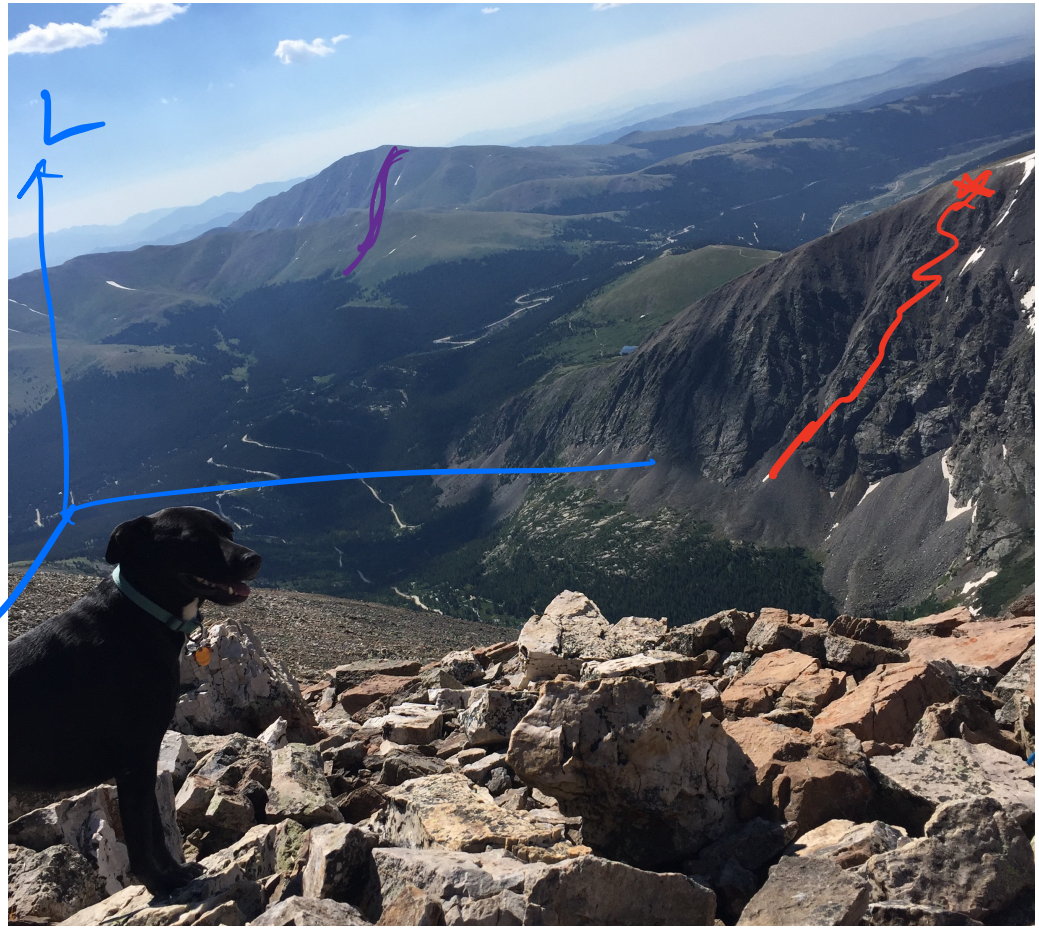
↳ σ can be anything

In DL, we call σ an activation function

How do we optimize something like this?



Optimization w/ gradient descent



Optimization w/ gradient descent

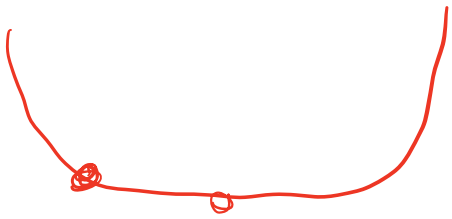
https://blog.skz.dev/gradient-descent

Optimization w/ gradient descent

Consider MSE loss, $\mathcal{L}(y,\hat{y}) = \|y-\hat{y}\|^2 = \|y - \vec{w}\cdot\vec{X}\|^2 = \mathcal{L}(\vec{w})$

fixed fixed

$$\mathcal{L}(\vec{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \vec{w}\cdot\vec{x}^i)^2 \quad (i \text{ is samples})$$



with two features:

$$\mathcal{L}(w_0,w_1) = \frac{1}{N}\sum\left[y_i - (w_0x_0^i + w_1x_1^i)\right]^2$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \frac{2}{N}\sum_{i=1}^{N}\left[y_i - (w_0x_1^i + w_1x_1^i)\right](-x_0^i)$$

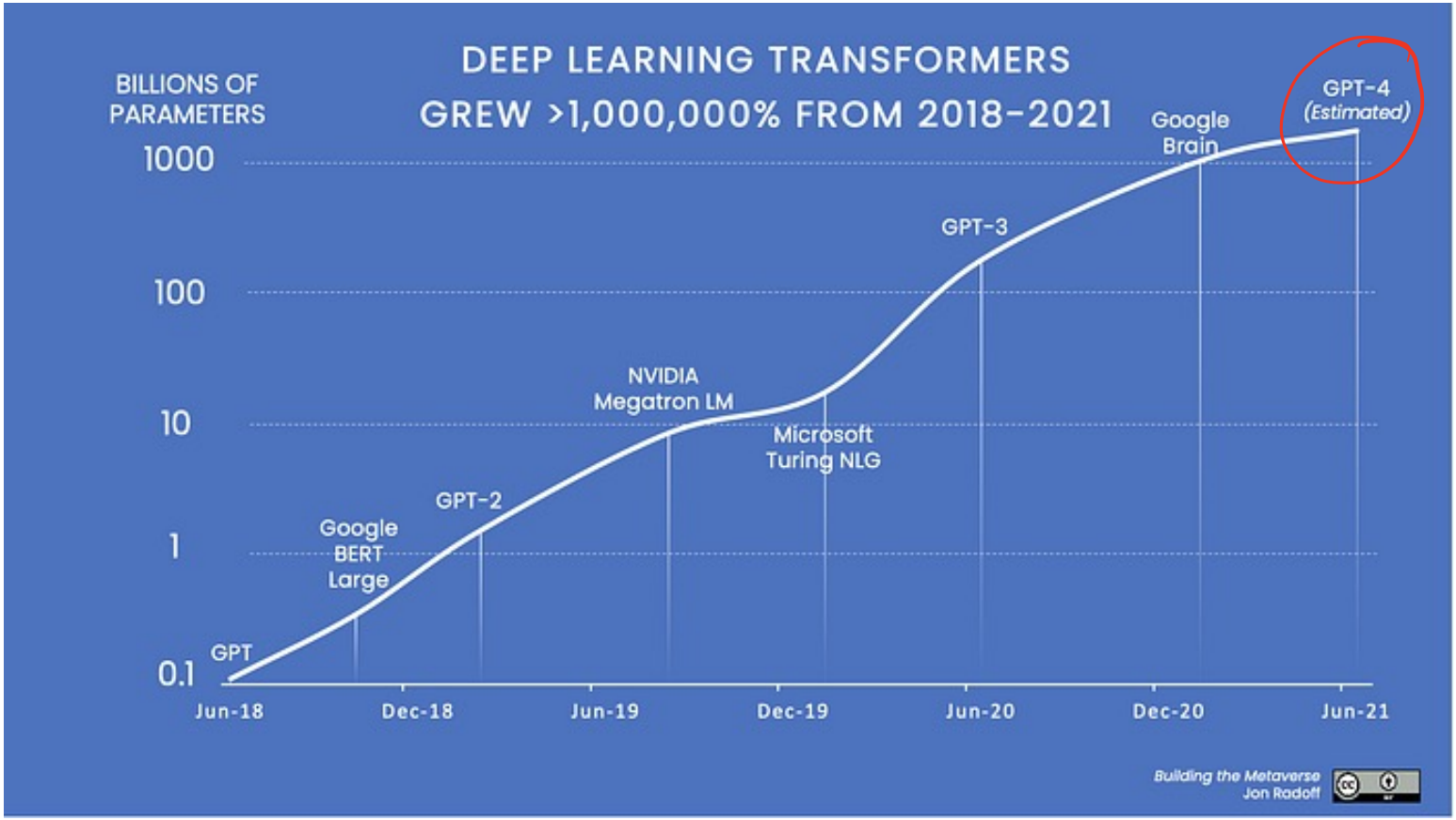
learning rate

$$w_0(1) = f(w_0(0)) = \ldots\ w_0(0) - \lambda \left.\frac{\partial \mathcal{L}}{\partial w_0}\right|_0$$

what would it I used a different loss functo?

Optimization w/ gradient descent

Neural networks have *lots* of weights



https://aryanbajaj13.medium.com/gpt-4-is-coming-soon-heres-what-we-know-so-far-gpt-gpt2-gpt3-43c3b38eeeb8

Even non-LLMs are pretty hefty

CEMS
Chemical Engineering
& Materials Science

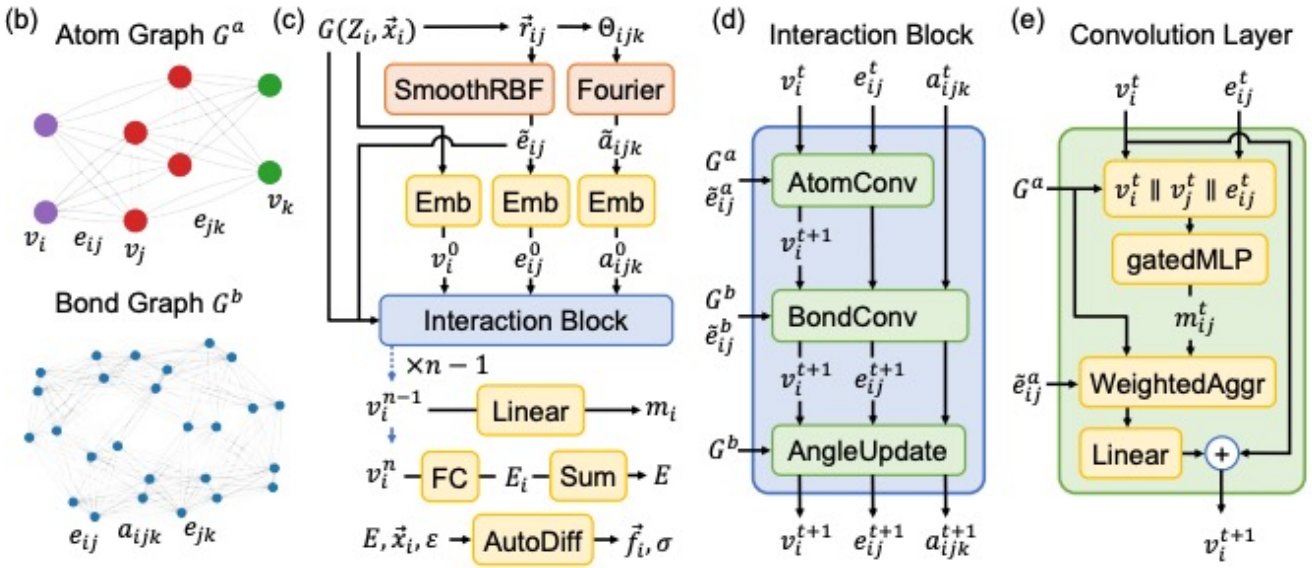
arXiv > cond-mat > arXiv:2302.14231

Condensed Matter > Materials Science

[Submitted on 28 Feb 2023]

CHGNet: Pretrained universal neural network potential for charge-informed atomistic modeling

Bowen Deng, Peichen Zhong, KyuJung Jun, Kevin Han, Christopher J. Bartel, Gerbrand Ceder



CHGNet with 400,438 trainable parameters was trained on the MPtrj dataset with 8:1:1 training, validation, and test set ratio partition by materials (see Meth-

exchange-correlation (see Methods). This effort resulted in a comprehensive dataset with 1,580,395 atom configurations, 1,580,395 energies, 7,944,833 magnetic moments, 49,295,660 forces, and 14,223,555 stresses. To ensure

In practice – stochastic, mini-batch, modified updates

Stochastic GD

$$\vec{w}(j+1) = \vec{w}(j) - \lambda \left.\frac{\partial \mathcal{L}^i(\vec{w})}{\partial \vec{w}}\right|_j$$

(one sample at a time)

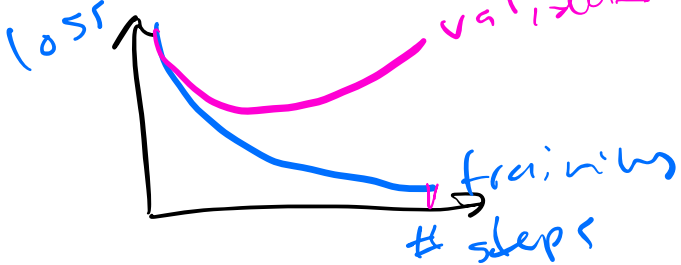
Mini-batch GD (a few samples a time)

Momentum:

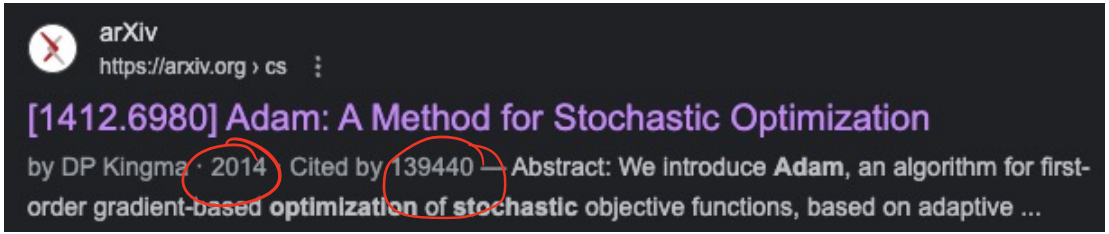
$$\vec{w}(j+1) = \vec{w}(j) - \lambda \left.\frac{\partial \mathcal{L}}{\partial w}\right|_j + \alpha \Delta w(j-1)$$

(λ: learning rate)

($\alpha > 1$: give me more momentum; $\alpha < 1$: friction)



A popular modified update method: Adam




```
moment1 = 0
moment2 = 0
for t in range(num_steps):
  dw = compute_gradient(w)
  moment1 = beta1 * moment1 + (1 - beta1) * dw
  moment2 = beta2 * moment2 + (1 - beta2) * dw * dw
  w -= learning_rate * moment1 / (moment2.sqrt() + 1e-7)
```

Adam

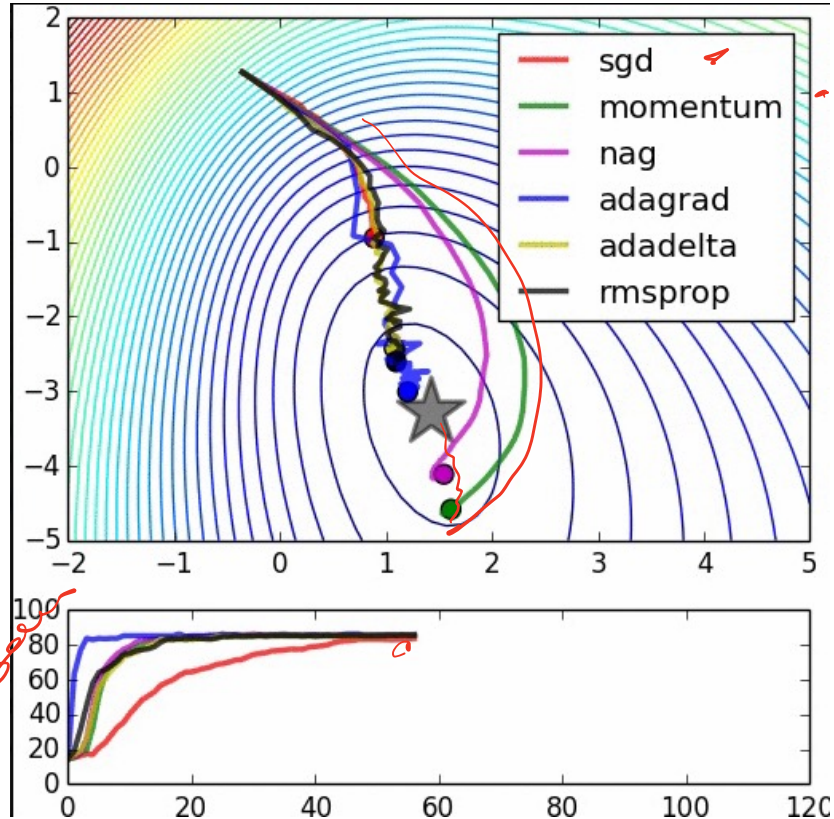
Momentum

AdaGrad / RMSProp

Bias correction

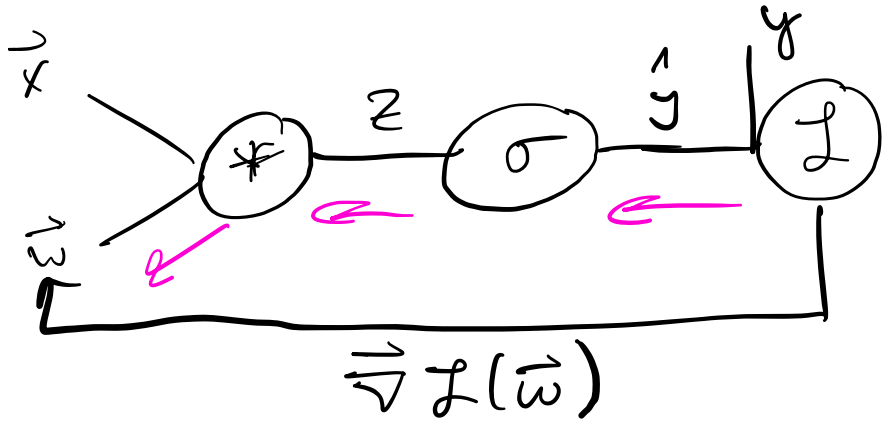
~72k hours since Adam was published
~140k citations 😲

Comparing optimizers

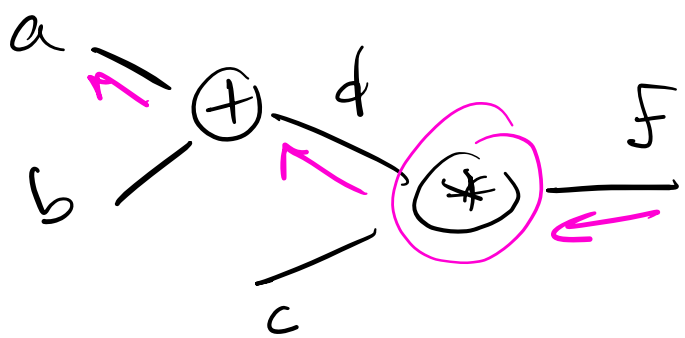


http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Doing this efficiently w/ backpropagation



Doing this efficiently w/ backpropagation



what is $\frac{\partial f}{\partial a}$?

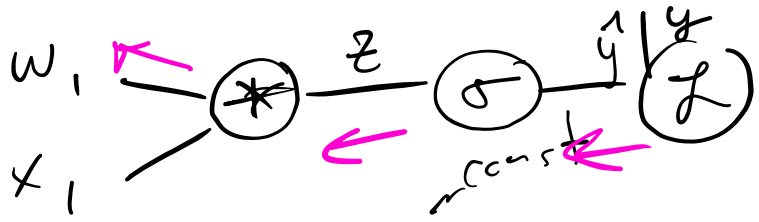
$\frac{\partial f}{\partial f} = 1$, $\frac{\partial f}{\partial d} = c$, $\frac{\partial d}{\partial a} = 1$

$f(a,b,c) = (a+b)c$

$\frac{\partial f}{\partial a} = (1)c(1) = c$

Linear model + linear activation

How does loss depend on weights?



$$\hat{y} = \sigma(z) = Az + b \quad b \sim \text{const}$$

$$\mathcal{L}(\hat{y}, y) = (y_i - \hat{y}_i)^2$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -2(y_i - \hat{y}_i)$$

$$= -2(y_i - (Az_i + b))$$

$$= -2(y_i - A(w_1 x_i + b))$$

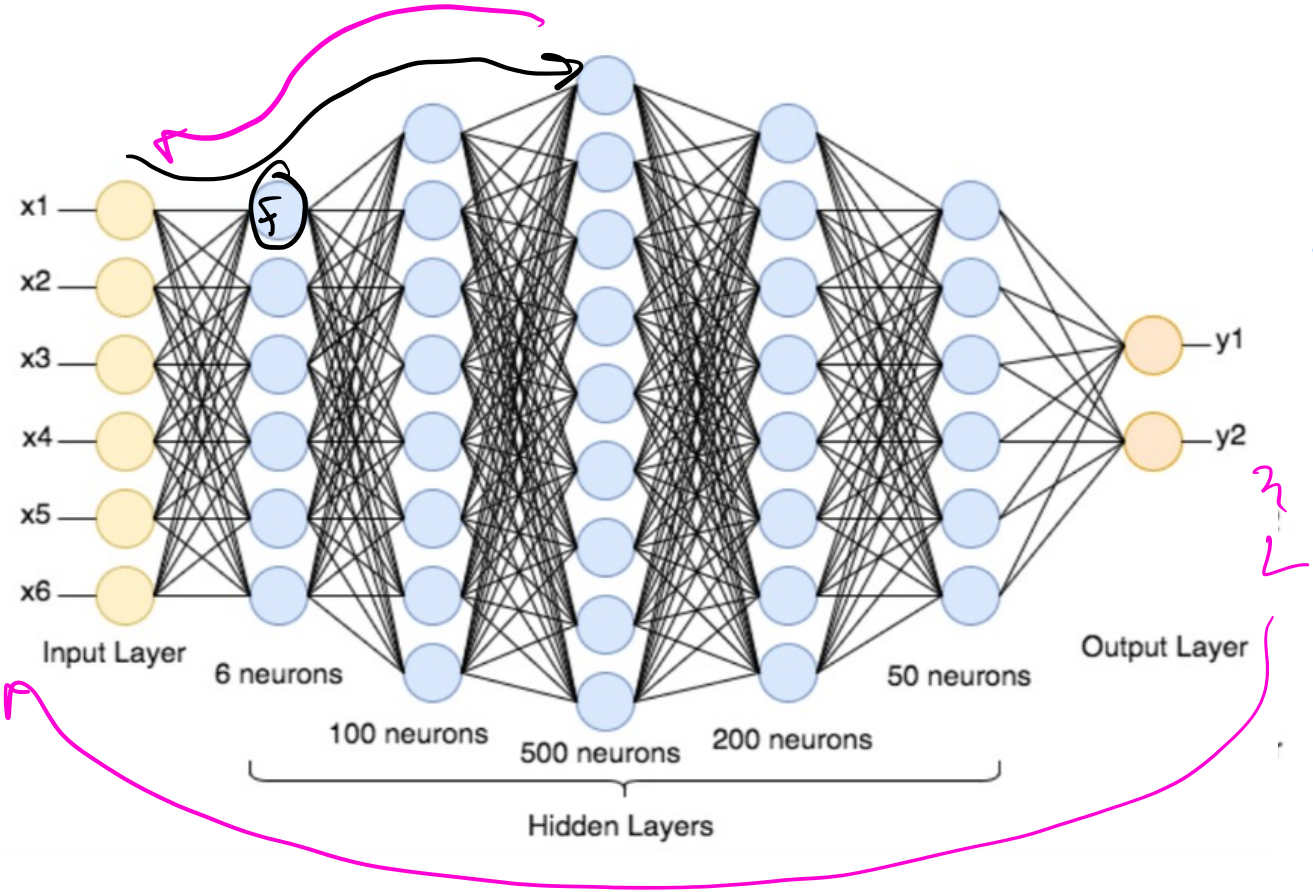
$$\frac{\partial \hat{y}}{\partial z} = A \quad , \quad \frac{\partial z}{\partial w_1} = x_1$$

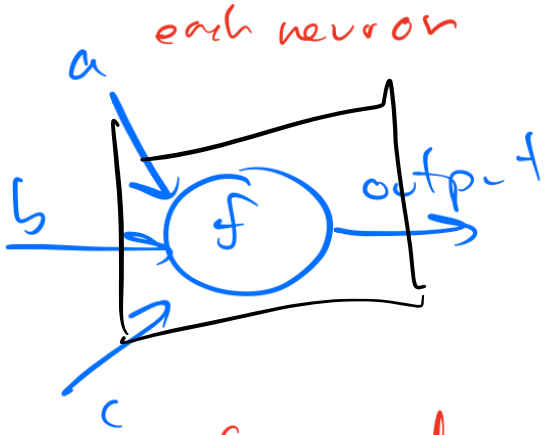
$$\frac{\partial \mathcal{L}}{\partial w_1} = -2Ax_1[y_i - A(w_1 x_i + b)]$$

JOIN CHIME

Go to:
https://chimein2.cla.umn.edu/join/608843
Or: visit chimein2.cla.umn.edu and enter
608-843

How this becomes a neural network





→ forward pass

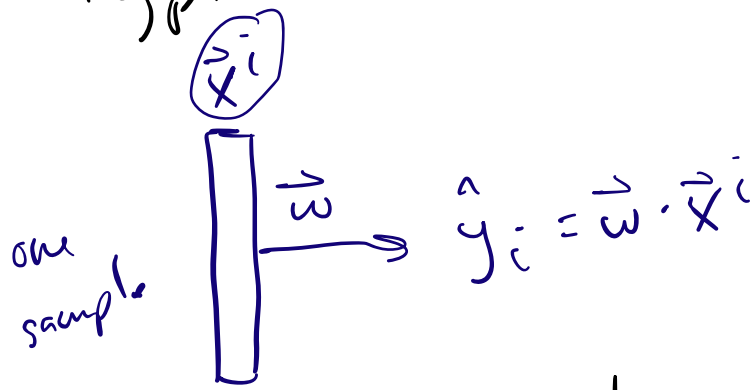
$f(a, b, c)$

← backward pass

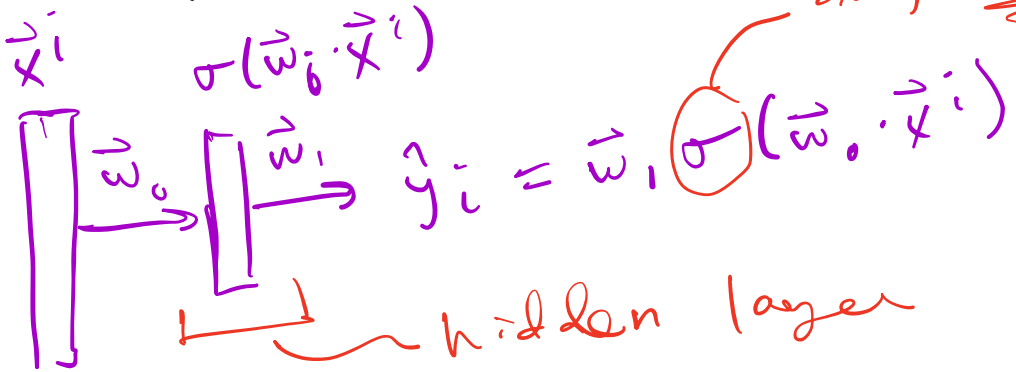
$\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}, \frac{\partial f}{\partial c}$

Activation functions make our NNs nonlinear

Typical linear model: $\hat{y} = f(\vec{x}) = \vec{w} \cdot \vec{x} = w_0 x_0 + w_1 x_1 + \dots$



Now w/ activation!

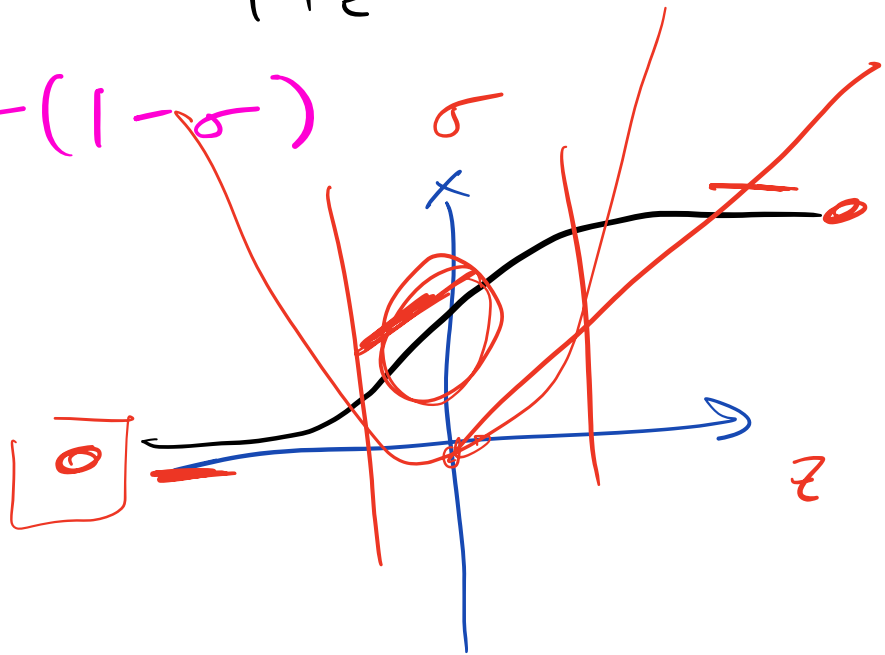


Vanishing gradients problem

Sigmoid activation

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial y}{\partial z} = \sigma(1-\sigma)$$



what happens when

1) $z \approx 0$, $\sigma = 1/2$, $\frac{\partial y}{\partial z} = \frac{1}{4}$

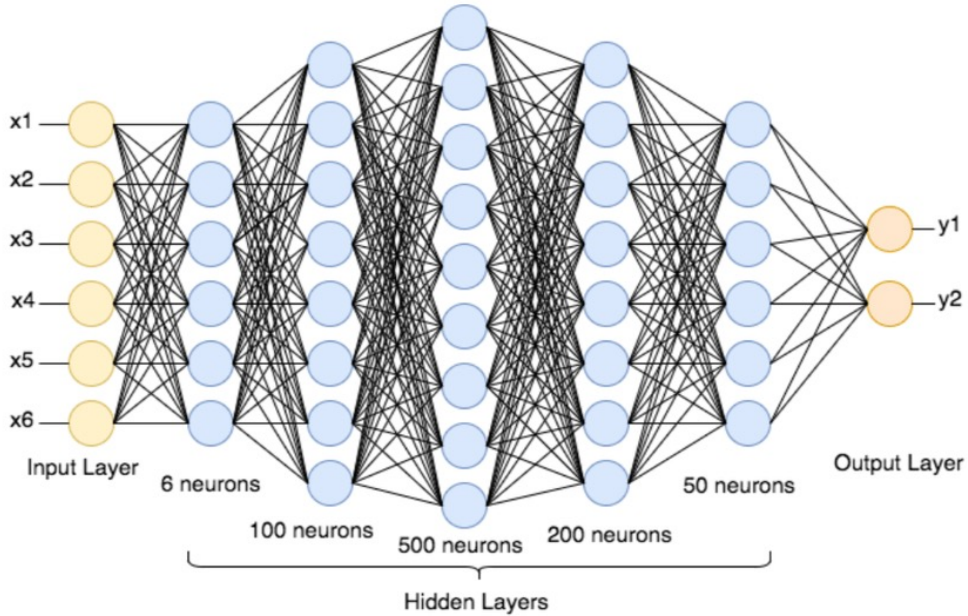
2) $z << 0$, $\sigma = 0$, $\frac{\partial y}{\partial z} = 0$

3) $z >> 0$, $\sigma = 1$, $\frac{\partial y}{\partial z} = 0$

Many forms of activation functions ...

https://atcold.github.io/pytorch-Deep-Learning/en/week11/11-1/

Normalizing features (~scaling) also matters



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

arXiv
https://arxiv.org › cs

Batch Normalization: Accelerating Deep Network Training ...

by S Ioffe · 2015 · Cited by 45718 — **Batch Normalization** allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, ...

As does initialization (of weights)

mlr.press
https://proceedings.mlr.press › ...

Understanding the difficulty of training deep feedforward ...

by X Glorot · Cited by 20132 — Our objective here is to understand better why standard gradient descent from random **initialization** is doing so poorly with deep neural networks, to better ...

8 pages

Xavier Initialization, or **Glorot Initialization**, is an initialization scheme for neural networks. Biases are initialized be 0 and the weights W_{ij} at each layer are initialized as:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

Where U is a uniform distribution and n is the size of the previous layer (number of columns in W).

Another knob to turn: regularization

LASSO & Ridge, $\hat{y} = f(\vec{x}) = \vec{w} \cdot \vec{x}$, $\mathcal{L} = \|y - \hat{y}\|^2 + \alpha \|\vec{w}\|^2$

L_2 is common for NN

(weight decay)

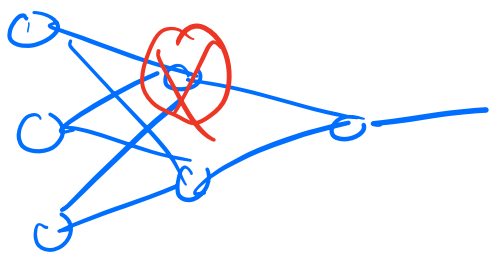
$\lambda = 1 \rightarrow$ LASSO

$\lambda = 2 \rightarrow$ Ridge

Case 1: $\vec{w} = [1,0,0,0]$, $\sum w_i^2 = 1+0+0+0$

2: $\vec{w} = [1/4, 1/4, 1/4, 1/4]$, $\sum w_i^2 = 4(1/16)$

Dropout



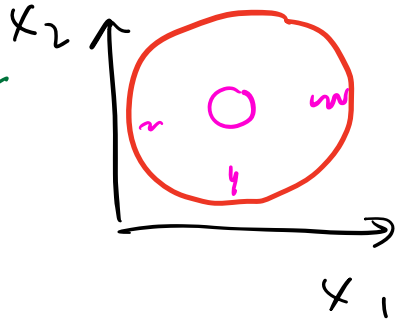
$$\frac{\partial \mathcal{L}}{\partial x_1} \quad \frac{\partial \mathcal{L}}{\partial x_2} \quad \frac{\partial \mathcal{L}}{\partial x_3}$$

Decisions to make when training

Network architecture
- # hidden layers
- size of hidden layers
- activation functions

Normalization
- Batch Norm

Initialization
- Xavier



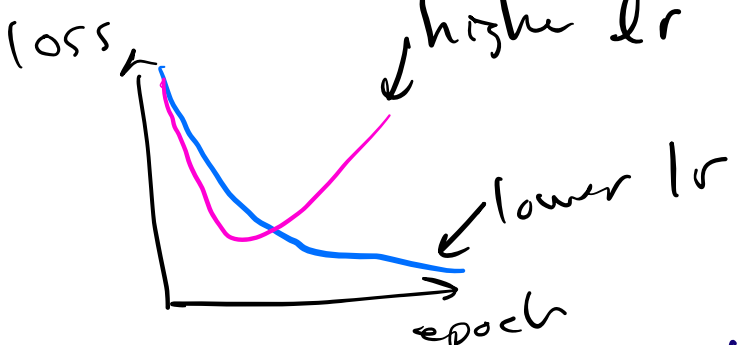
Optimization
- Adam
- learning rate

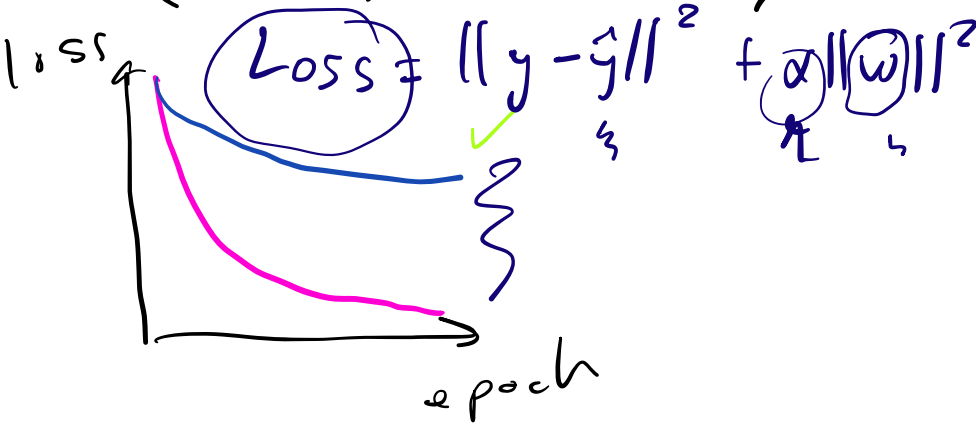
Regularization
- L_2
- Dropout

How long to train
- # epochs
- early stopping criteria
- batch size

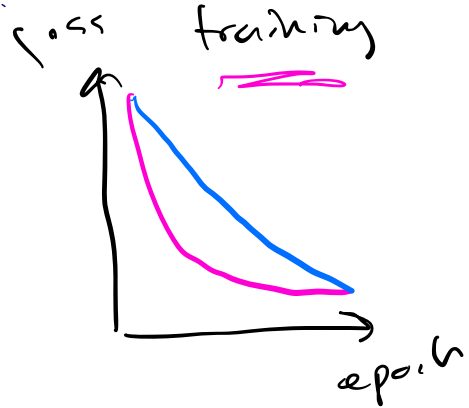
Let's look at some loss curves

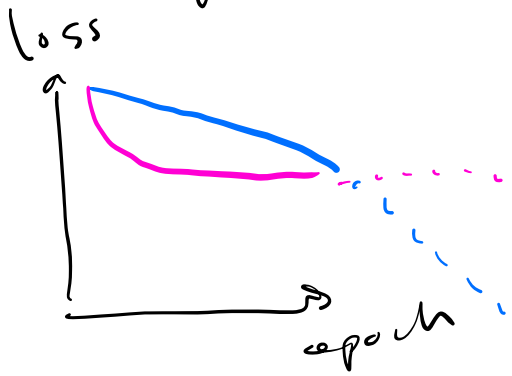
(1) Which has higher learning rate? (2) higher regularization?





(3) Which model looks better





JOIN CHIME

Go to:
https://chimein2.cla.umn.edu/join/608843
Or: visit chimein2.cla.umn.edu and enter
608-843

Let's see how these things work

https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html