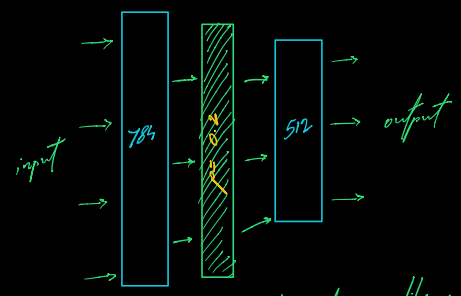
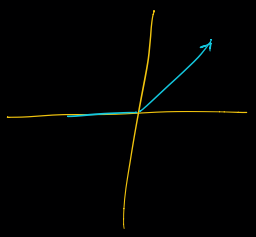
for every weight we multiply it with "p" during run time.



↳ dropout is like a layer in-between. That's how we put it in Keras or TF.

⇒ ReLu → Rectified Linear Unit.

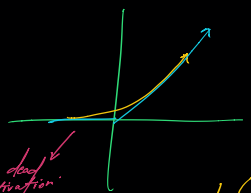
↳ $R(z) = \max(0, z)$



$$\frac{\partial R(z)}{\partial z} = \begin{cases} 0 & \text{if } z \leq 0 \\ z & \text{otherwise.} \end{cases}$$

Even though this is not differentiable, we don't give a fuck, we just use a hack to make it work. ⟶ ReLu is too good compared to tanh or σ

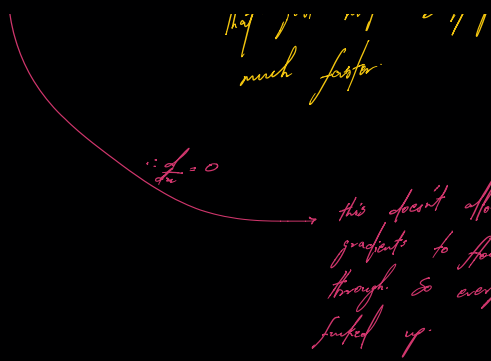
It converges a lot faster than them. ∴ no vanishing gradient.



~~softplus~~ → $\log(1 + e^z)$

~~ReLu~~

→ for ReLu, it's basically NO CALCULATION for getting the $\frac{\partial}{\partial z}$. It just makes backprop so



this doesn't allow gradients to flow through. So everything fucked up.

$\Rightarrow$ That's why Leaky ReLU. $\rightarrow \begin{cases} 0.001z & \text{if } z \le 0 \\ z & \text{otherwise} \end{cases}$

↳ doesn't become zero.

Initialize the weights → random → normal distribution.

$w_{ij}^k \neq 0$ → this creates fuck up.

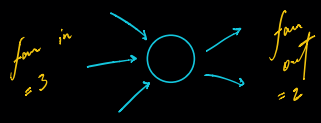
① → All other activations in further layers will be zero.

② Symmetric. So each weight will have same gradient.

③ init should not have too large or too small. ⇒ Bad gradients.

→ for ensembles, the more different the base models are, the better the outcome.
which is why asymmetry is important with init.

It's also important to normalize the datasets itself. to prevent exploding / vanishing gradients.



works well for σ ← $w_{ij}^k \sim$ Uniform Random $\left[\frac{-1}{\sqrt{f_{in}}}, \frac{+1}{\sqrt{f_{out}}}\right]$

this is all empirical, with little theoretical support behind it.

Xavier / Glorot init.

(a.) Normal

$w_{ij}^k \sim N(0, \sigma)$

$\sigma = \frac{2}{f_{in} + f_{out}}$

(b.) Uniform

$w_{ij}^k \sim \text{Uniform}\left[\frac{-\sqrt{6}}{\sqrt{f_{in} + f_{out}}}, \frac{+\sqrt{6}}{\sqrt{f_{in} + f_{out}}}\right]$

↳ "Xavier initialization"

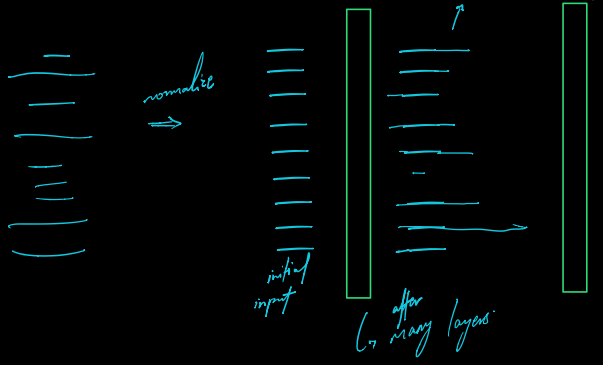
He init → works well with ReLU

(a) Normal $w_{ij}^k \sim N(0, \sigma)$

$$\sigma = \sqrt{\frac{2}{f_{in}}}$$

(b) Uniform $w_{ij}^k \sim U\left[-\sqrt{\frac{6}{f_{in}}}, +\sqrt{\frac{6}{f_{in}}}\right]$

Batch Normalization.



scale changed AF for each batch each time.
↳ not good for this layer.
↳ it will see a completely different distribution each batch.
fucked up.

→ Batch normalization means norms not only with input but also after every layer.

standardize: $\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ (μ → mean; ε → to prevent divide by zero)

$x_i = \gamma \hat{x}_i + \beta$

↳ these two are learned.

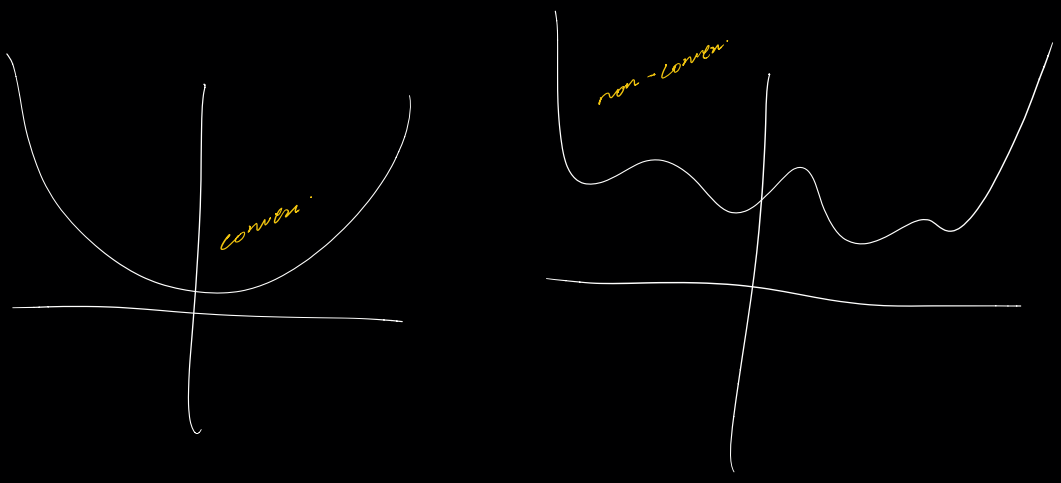
This thing happens for each feature during a batch.

BN also works as a weak regularizer.

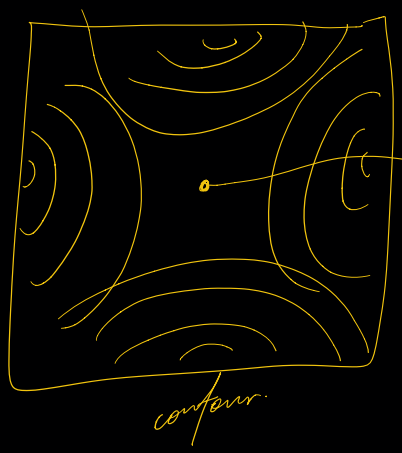
It prevents internal covariate shift and thus allows training deeper neural networks.

Convex f^n. → has only one minima.
↳ LR, SVM, etc.
local minima = global minima

Non convex f^n. → multiple local minima.
↳ Deep Learning



have to be careful not to settle on a local minima.



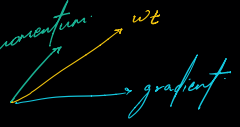
Momentum with SGD.

$v_1 = \eta g_1$

$v_t = \gamma v_{t-1} + \eta g_t$

↳ $0 \le \gamma \le 1$

↳ typically 0.9



$w_t = w_{t-1} - [\gamma v_{t-1} + \eta g_t]$

↳ here we take the addition of momentum & grad

↳ both are applied together.

Nesterov Accelerated Gradient (NAG)

→ here we apply momentum first, then compute grad at that point(w') then apply it.

↳ momentum applied first & then grad

$w_t = w_{t-1} - (\gamma v_{t-1} + \eta g')$

$g' = \left(\frac{\partial L}{\partial w}\right)_{w'}$

$w' = w_t - \gamma v_{t-1}$

NAG has kinda the same performance as SGD + momentum. Nothing fancy.

Ada. Grad.

→ Adaptive learning rate

$$w_t = w_{t-1} - \eta'_t g_t$$

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_{t-1} + \varepsilon}}$$

η → 0.01 whatever you give

↳ even η is time dependent now.

↳ just to avoid divide by zero

$$\alpha_{t-1} = \sum_{i=1}^{t} g_i^2$$

↳ all grads up to this point.

↓ always positive

→ this keeps on ↑ and thus η'_t reduces with time.

(+) → No need to fiddle with η

(+) → sparse features benefit from it

(−) → η does ↓ but then later on it gets super low as t ↑

↓ slow convergence

Ada Delta & RMS Prop

$$w_t = w_{t-1} - \eta'_t g_t$$

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_{t-1} + \varepsilon}}$$

$$\alpha_{t-1} = \left[\gamma \alpha_{t-2} + (1-\gamma) g_{t-1}^2 \right]$$

↓ normally 0.9

$$\alpha_{t-1} = 0.1\, g_{t-1}^2 + 0.9 \left[0.1\, g_{t-2}^2 + 0.9 \left[\alpha_{t-3} \right] \right]$$

↳ So on...

→ We're controlling the growth of denominator

→ exp. weighted avg of g_i^2

→ instead of sum of g_i^2

→ The upsides still remain from Ada Grad

Adam

[Adaptive Moment Estimation]

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad 0 \le \beta_1, \beta_2 \le 1$$

(β_1: 0.9 [generally])

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

(β_2: 0.99)

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t} \qquad \hat{v}_t = \frac{v_t}{1-(\beta_2)^t}$$

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

Monitoring $\frac{\Delta w}{w}$ of layers. → should do it for each layer, epoch → good practice.

Vanishing / exploding gradients can be detected.

↳ can be avoided using gradient clipping. → $g_{new} = \left(\frac{g}{\|g\|_2}\right) * z$ (threshold)

↳ $< 1 * 1$

∴ < 1

All libraries have this L2 norm clipping feature.

Softmax

↳ only works for binary classification.

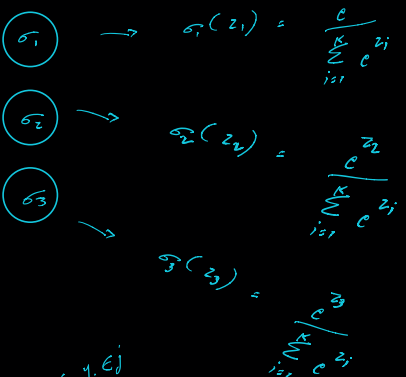
we want the probs to add up to 1

we want multiclass

This just an extension of LR

↳ generalization.

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



N pts

K class

$$\frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \log p_{ij}$$

$y_{ij} = 1$ if $y_i \in j$

For every point

& for every class we get log loss.

→ This is an extension of log loss

This is for classification,

for regression we make it a linear unit

so $f(z) = z = w^T x$

And loss is squared loss.

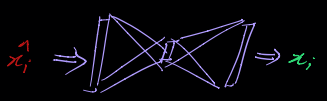
How to Train an MLP (Summary)

1. Preprocess → Normalize
2. weight init → Xavier/Glorot → Sigmoid/Tanh
   → He → ReLu
   → Gaussian with small σ
3. Activation → ReLU
4. BN and Dropout
5. Optimizer → Adam #Best
6. Hyperparameters → #layers
   # neurons
   # dropout rate
7. Loss → squared loss → regression.
   → multiclass logloss → classification
8. Monitor gradients → Clipping
9. Not using tensorboard → 
10. Avoid overfitting.

Even missing one of these points is enough to throw us off

Denoising AutoEncoder

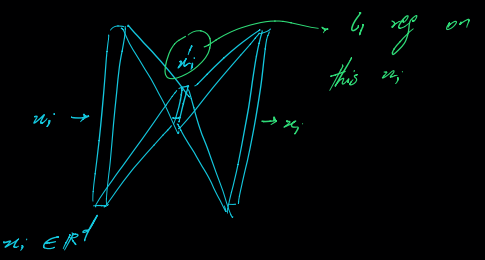
$$\hat{x_i} = x_i + N(0, \sigma)$$



Data is intentionally corrupted and AE is trained to get pure data from slightly corrupted data.

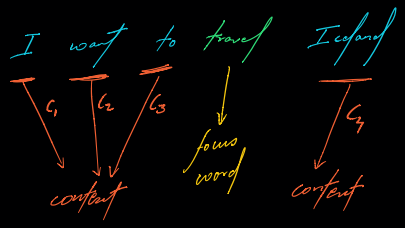
Sparse AE

if feature vector is too sparse



If linear activations are used and only one sigmoid hidden layer is taken, the result is kinda similar to what PCA would give us.

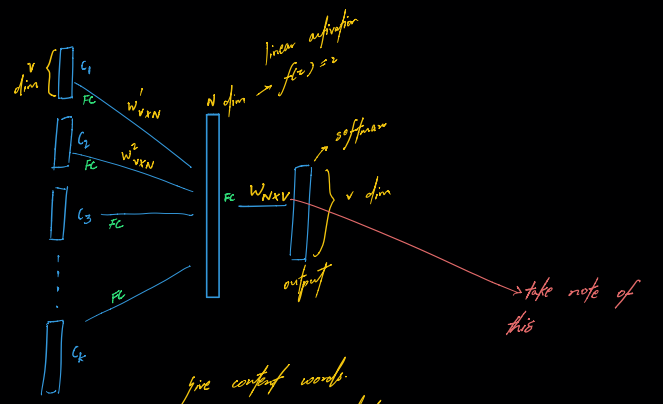
Word2Vec → not a deep learning algo.



CBOW
↳ Collectiol Bag of Words.

① V = size of vocab

② one hot encode each word
$w_i \in \mathbb{R}^V$

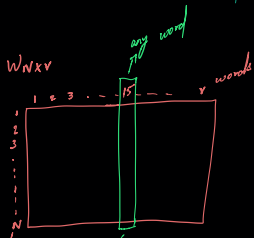


give context words.
see if it can predict focus word.

Training data.

focus context

travel { I, want, to, Iceland }



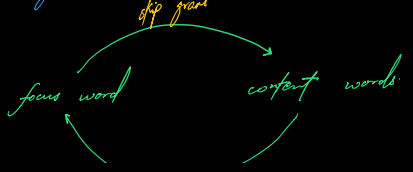
↳ this vector is what we mean by getting word2vec

Dimension of hidden layer. Can be changed depending on need.

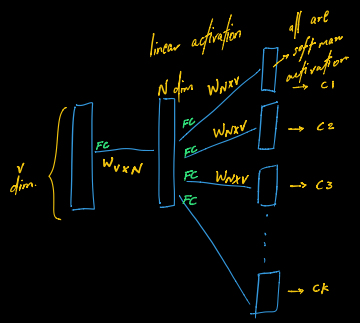
Also the Dimension of w2vector.

Skipgram:

↳ small changes to CBOW



CBOW



The no. of weights are same with CBOW & skipgram. $(k+1)(N \times V)$

#quick easy: context given → predict one word

#hard one word given → predict context

↳ : more powerful

CBOW → 1 softmax → fast → good for freq. words. coz we'll have more context words

skipgram → k softmax → slow → works with less data and

k → # context words.

$k\uparrow$ → more context → better results

N → # dim.

$N\uparrow$ → more information.

weights $= (k+1)(N \times V)$

$N = 200$ dim

$k = 5$

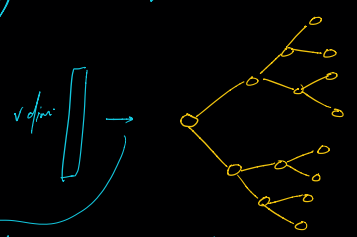
$V = 10k$ words

Still # weights = 12 million.

#Need to optimize!!

Hierarchical Softmax

optimize V-softmax somehow.



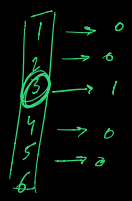
↳ Make the softmax predict the path of v leaf node. target

Still need more clarification "why" this works.

So we converted $O(v)$ to $O(\log_2 v)$

Optimization 2

Negative Sampling



→ always update this one.

+

for the others we have a probability that they will be updated.

generally 10^{-5}

$$P(w_i) = 1 - \sqrt{\frac{t}{freq.(w_i)}}$$

This is only intuition for hierarchical softmax and negative sampling.