Advanced Machine Learning, 22 Aug 2019

Feedforward NN — nonlinearity

Cross entropy vs log likelihood

Output activation functions

z

$\sigma(z) = \frac{1}{1+e^{-z}}$ 0-1 output

Multiway classification $z_1, z_2, \ldots, z_K$

$\text{Softmax}_{\text{argmax } j} = \frac{\exp(z_j)}{\sum \exp(z_i)}$ Softargmax

Crossentropy $\log P(z_i|x)$

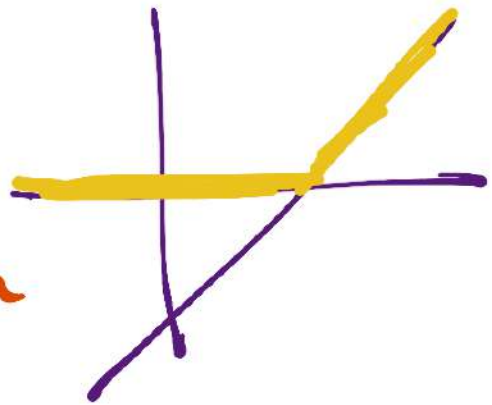
$$\log \frac{e^{z_j}}{\sum e^{z_i}} = z_j - \log \sum e^{z_i}$$

Interior nodes – nonlinearity

$$x \to \phi(x) \to z \to f(z)$$

RELU

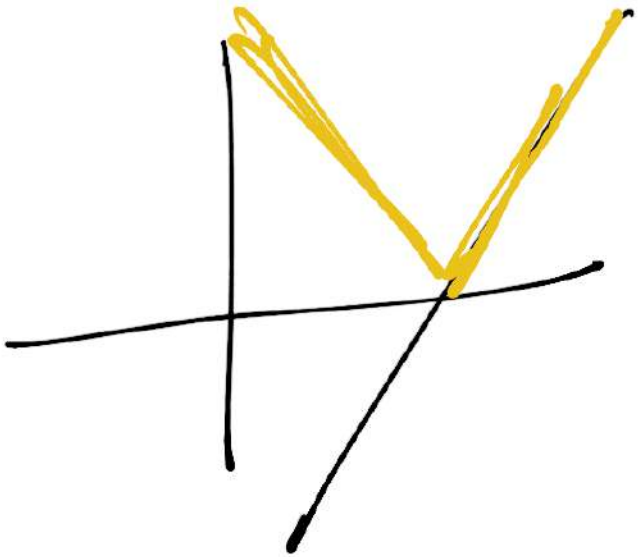
Rectified Linear Unit



$\max(0, z)$

Generalizations to handle zero gradient

Absolute value rectification



More generally

$$|z| = \max(0,z) - \min(0,z)$$

$\alpha = -1$

$$\max(0,z) + \alpha \min(0,z)$$

Architecture

How many layers? Depth

How many nodes per layer? Width

How are the layers connected?

└ Assume completely connected

Recall:

Can construct a 1 hidden layer network to approximate any f using steps

Theoretically, 1 hidden layer suffices

- Width explodes
- Parameters explode

Trade off in using more layers is to reduce width per layer

Adding layers reduces parameters

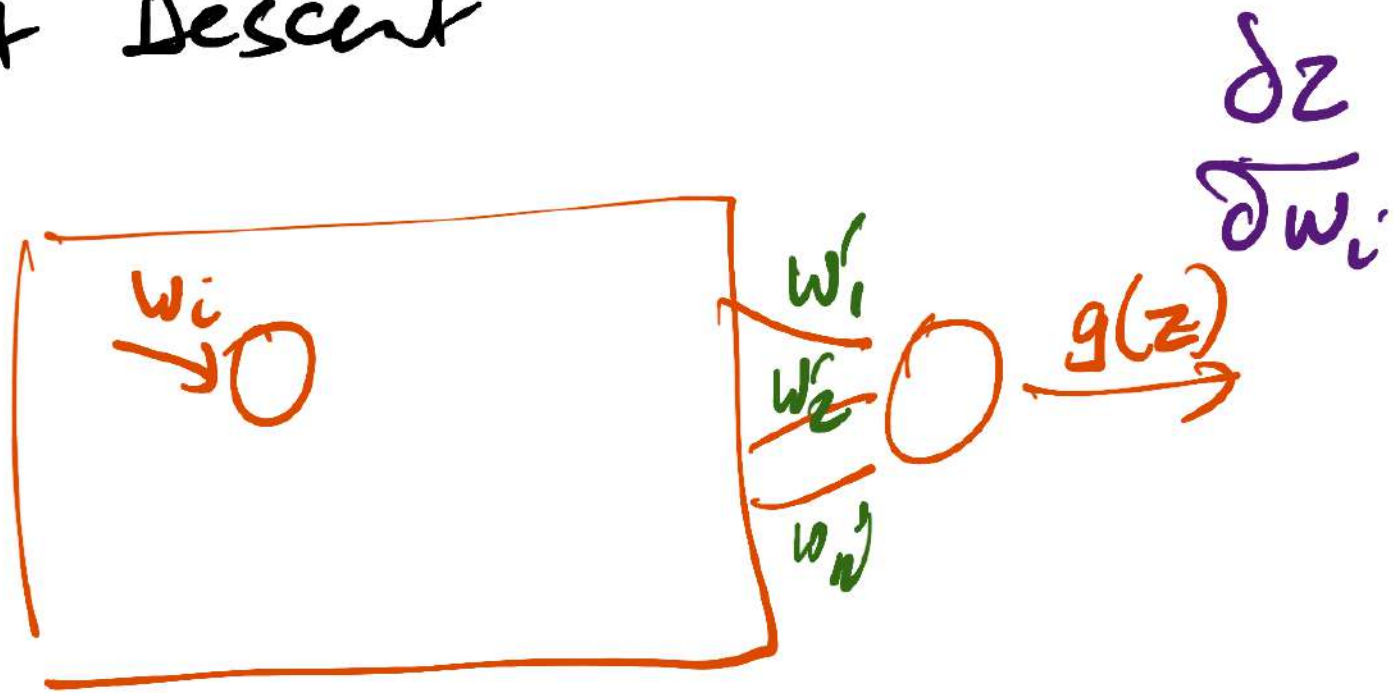
Each layer is an abstract "feature"

How many layers, how wide?

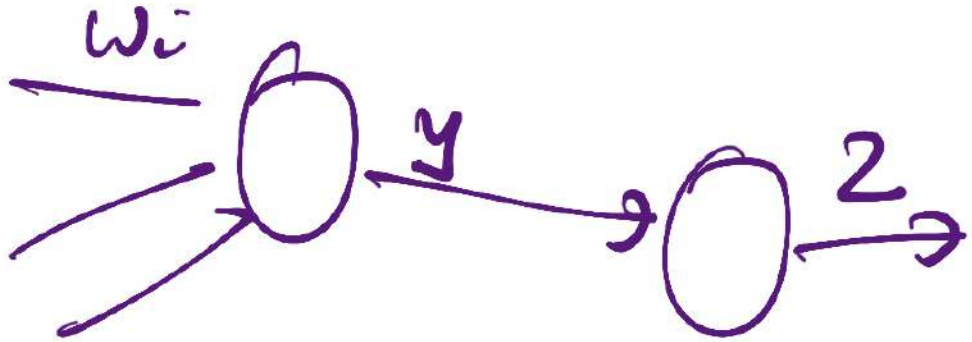
Empirical

Choose a value that you can compute!

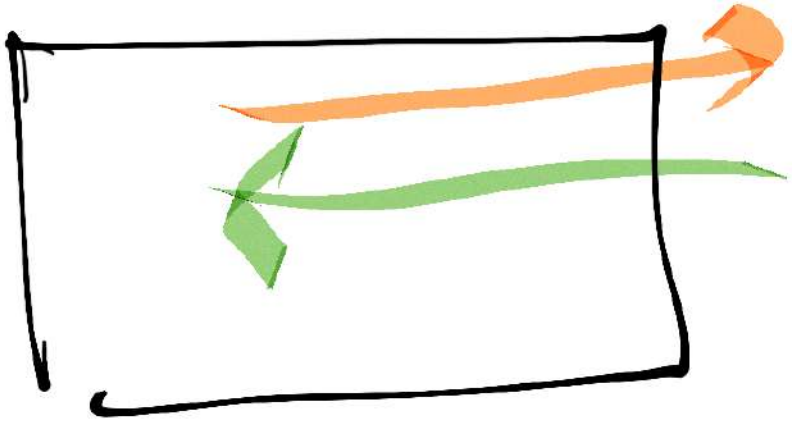
Gradient Descent



Chain rule



$$\frac{\partial z}{\partial w_i} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial w_i}$$

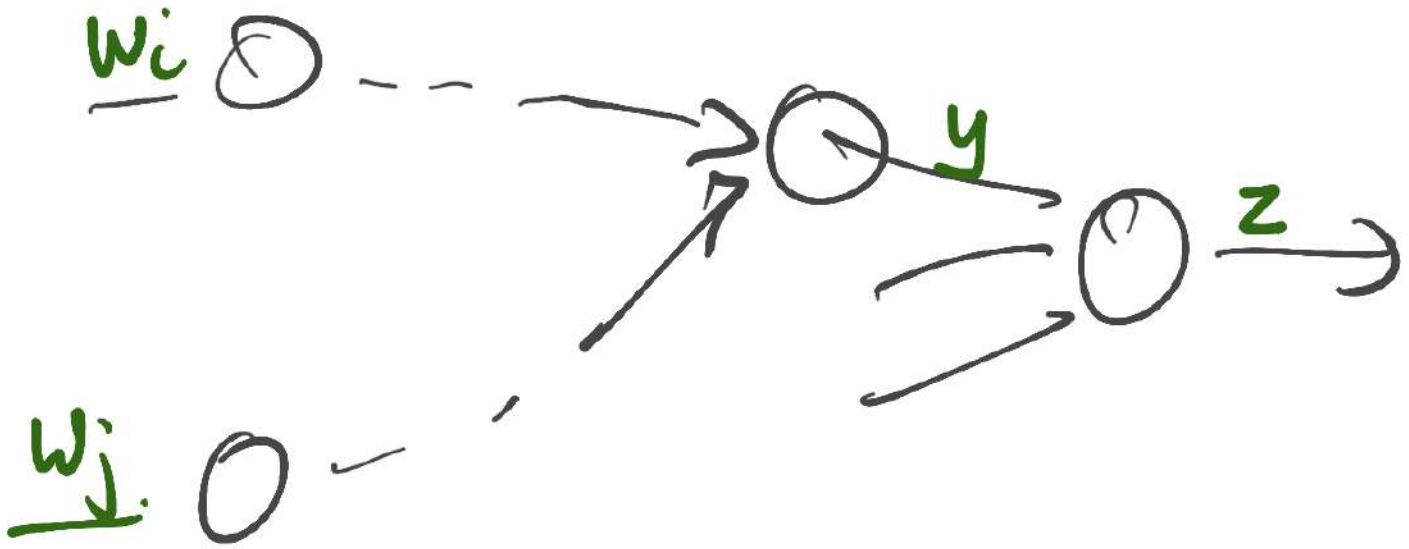


Compute outputs forward

Compute gradients backwards

Backwards calculation exploits the fact that the network is a directed acyclic graph

Keep calculated values stored - Dynamic Programming



Backpropagation can be done

- Numerically — Torch
- Symbolically — Tensorflow

- Loss function
- Activation function
- Structure of network
- Gradient descent

Optimizing the process of learning

1. Tune the learning to generalize better

2. Speed it up

1. Regularization

   Add a cost based on model complexity

   Postpone / Skip

2. Speed up the computation

Minimizing loss w.r.t training data

$$E_{(x,y)\sim \hat{p}_{data}} L(f(x;\theta), y)$$

Loss function, parameter θ, input x, real output y

What we really want

Minimizing risk

$$E_{(x,y)\sim p_{data}} L(f(x;\theta), y)$$

Natural loss function is 0-1 loss

0 if $f(x) = y$

1 if $f(x) \neq y$

↓ "surrogate" loss functions

log likelihood

Gradient descent

Most cases, loss function is not convex

Batch - take mean over entire input

Single input - adjust per input

Mini batches - Stochastic Gradient Descent

Statistically

Sample of size n

True mean μ Variation $\frac{\sigma}{\sqrt{n}}$

Important to choose mini batches as random samples

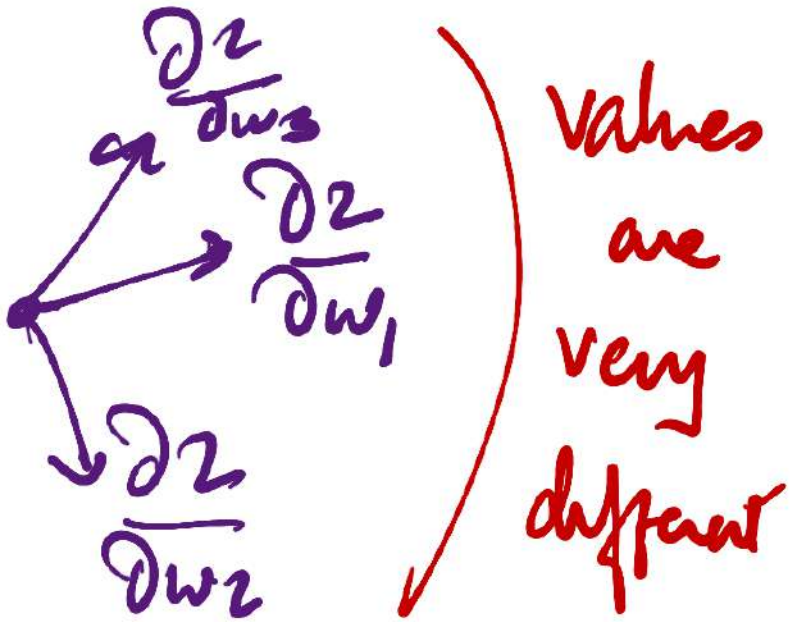
Issues

1. Multiple minima

2. Plateaus

3. Ill-conditioning

Learning rate ε

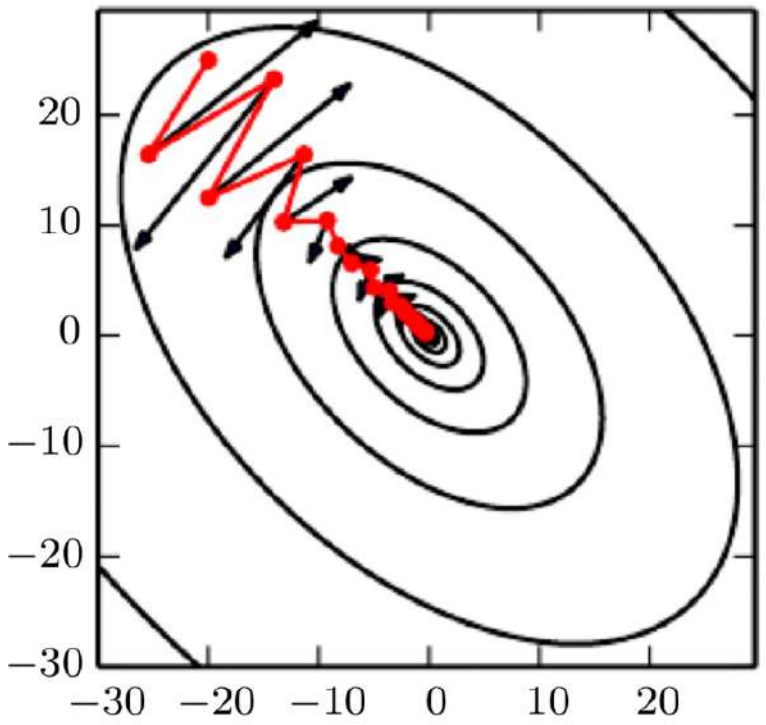


Add a parameter called "momentum"

Adjusting ε

Rate increases if direction does not change.

Rate decreases if direction keeps changing.



Variations

Nesterov momentum

Hyperparameter - Weight of the momentum term

Directly adapt ε

Adaptive learning rate

Delta-bar-delta — Decrease ε if sign changes
Increase ε otherwise

Delta-bar-delta

- sign wrt entire history

Narrow the history to a recent window

AdaGrad $-\frac{1}{\sqrt{\text{sum of all gradients}}}$

Bigger past values give bigger changes vs ϵ

RMSProp - exponential decay ↑
Delta-bar-Delta

Adam - Adaptive moments

RMSProp + momentum

Initialization

Black magic

Identical hidden nodes - different values

Assign randomly

- Normal
- Uniform

Use small weight