MACHINE LEARNING

Deep Neural Networks: Neural Networks with Tabular Data

Last Update: 30th November 2022

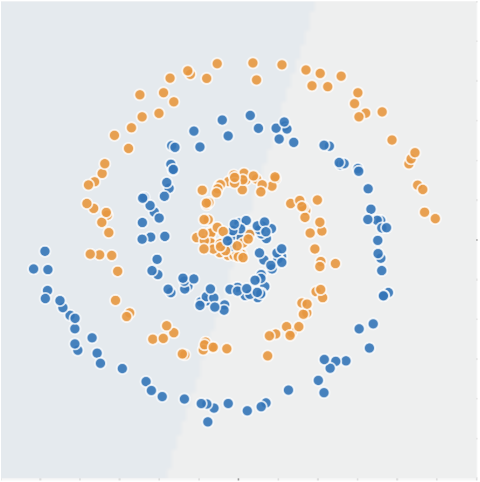
Prof. Dr. Shadi Albarqouni

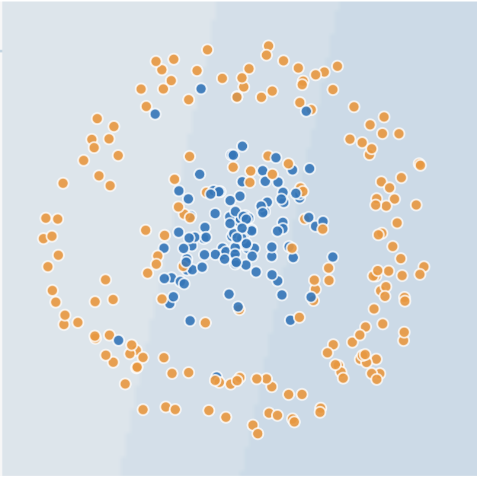
Director of Computational Imaging Research Lab. (Albarqouni Lab.)
University Hopsital Bonn | University of Bonn | Helmholtz Munich

STRUCTURE

BASIS FUNCTION EXPANSION

NONLINEAR CLASSIFIER





The non-linear transofmation $\phi(x)$ can be spedificed by hand; which is very limiting,

$$f(x;\theta) = w^T\phi(x) + b \quad \text{where} \quad \theta = (w, b)$$

The non-linear transofmation $\phi(x)$ can be spedificed by hand; which is very limiting,

$$f(x;\theta) = w^T\phi(x) + b \quad \text{where} \quad \theta = (w, b)$$

A natural extension is to have a feature extractor with a new set of parameters;

$$f(x;\theta) = w^T\phi(x;\theta_2) + b \quad \text{where} \quad \theta = (\theta_1, \theta_2) \quad \text{and} \quad \theta_1 = (w, b)$$

The non-linear transofmation $\phi(x)$ can be spedificed by hand; which is very limiting,

$$f(x;\theta) = w^T\phi(x) + b \quad \text{where} \quad \theta = (w, b)$$

A natural extension is to have a feature extractor with a new set of parameters;

$$f(x;\theta) = w^T\phi(x;\theta_2) + b \quad \text{where} \quad \theta = (\theta_1, \theta_2) \quad \text{and} \quad \theta_1 = (w, b)$$

To create more and more complex functions; one can repeat the same process multiple times recursively,

$$f(x;\theta) = f_L(f_{L-1}(\dots(f_1(x;\theta_1)\dots);\theta_{L-1});\theta_L)$$

This is the key idea behind deep neural networks (DNNs). This is known as a feedforward neural network (FFNN).

MULTILAYER PERCEPTRONS (MLPS)

MOTIVATION

Example: XOR Perceptron challenge

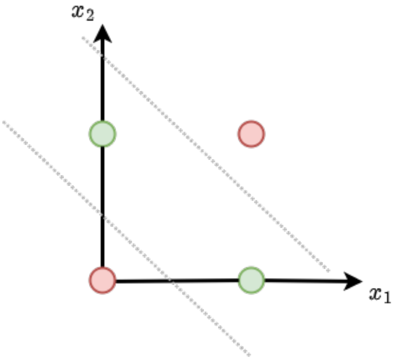
Heaviside function | Linear classifier | Perceptron

Perceptron:

$$
\begin{aligned}
H(a) &= H(w^T x + b) \\
&= \mathbb{I}(w^T x + b > 0)
\end{aligned}
$$

XOR Table:

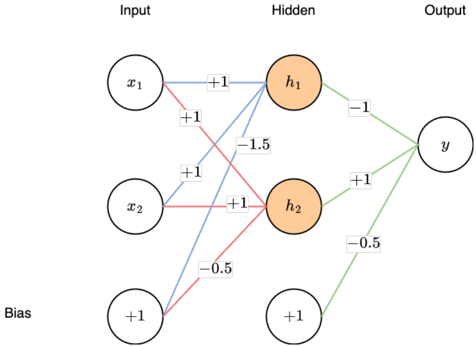
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



MOTIVATION

Example: Challenge accepted -- Multilayer Perceptron (MLP)

Heaviside function | Linear classifier | Perceptron



$$
\begin{aligned}
h_1 &= x_1 + x_2 - 1.5 \triangleq x_1 \wedge x_2 \\
h_2 &= x_1 + x_2 - 0.5 \triangleq x_1 \vee x_2 \\
y &= -h_1 + h_2 - 0.5 \\
y &= \overline{(x_1 \wedge x_2)} \wedge (x_1 \vee x_2)
\end{aligned}
$$

Revisit the previous slide and show the hyperplanes h_1, h_1 and y

MULTILAYER PERCEPTRONS (MLPS)

Multilayer perceptrons (MLPs)

A multilayer perceptron (MLP) is a stack of perceptrons, each of which involved the non-differentiable Heaviside function. This makes such models difficult to train, which is why they were never widely used. MLP is also defined as a fully connected class of feedforward neural network.

To make the MLPs differentiable, we replace the Heaviside function $H : \mathbb{R} \rightarrow \{0, 1\}$ with a differentiable activation function $\psi : \mathbb{R} \rightarrow \mathbb{R}$.

$$\mathbf{h_l} = f_l(\mathbf{h_{l-1}}) = \psi_l(\mathbf{b_l} + \mathbf{W_l h_{l-1}}) = \psi(\mathbf{a_l})$$

where $\mathbf{a_l}$ is the pre-activations and $\psi(\cdot)$ is the activation function

MULTILAYER PERCEPTRONS (MLPS)

Multilayer perceptrons (MLPs)

A multilayer perceptron (MLP) is a stack of perceptrons, each of which involved the non-differentiable Heaviside function. This makes such models difficult to train, which is why they were never widely used. MLP is also defined as a fully connected class of feedforward neural network.

To make the MLPs differentiable, we replace the Heaviside function $H : \mathbb{R} \to \{0, 1\}$ with a differentiable activation function $\psi : \mathbb{R} \to \mathbb{R}$.

$$\mathbf{h_l} = f_l(\mathbf{h_{l-1}}) = \psi_l(\mathbf{b_l} + \mathbf{W_l h_{l-1}}) = \psi(\mathbf{a_l})$$

where $\mathbf{a_l}$ is the pre-activations and $\psi(\cdot)$ is the activation function

This can be written in a scalar form as

$$h_{kl} = \psi_l \left(b_{kl} + \sum_{j=1}^{K_{l-1}} w_{jkl} h_{jl-1} \right)$$

ACTIVATION FUNCTIONS

Linear functions? –No, this results in a simple linear classifier

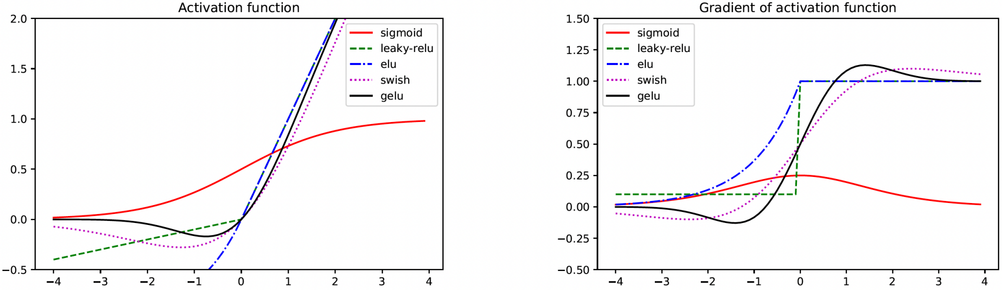
$$f(x;\theta) = W_L(W_{L-1}(\dots(W_1 x)\dots)) = W_L W_{L-1}\dots W_1 x = Wx$$

Alternatives

Name	Definition	Range	Reference
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$[0,1]$	
Hyperbolic tangent	$\tanh(a) = 2\sigma(2a) - 1$	$[-1,1]$	
Softplus	$\sigma_+(a) = \log(1+e^a)$	$[0,\infty]$	[GBB11]
Rectified linear unit	$\mathrm{ReLU}(a) = \max(a,0)$	$[0,\infty]$	[GBB11; KSH12]
Leaky ReLU	$\max(a,0) + \alpha\min(a,0)$	$[-\infty,\infty]$	[MHN13]
Exponential linear unit	$\max(a,0) + \min(\alpha(e^a - 1),0)$	$[-\infty,\infty]$	[CUH16]
Swish	$a\sigma(a)$	$[-\infty,\infty]$	[RZL17]
GELU	$a\Phi(a)$	$[-\infty,\infty]$	[HG16]

ACTIVATION FUNCTIONS

Why ReLU (Leaky-ReLU) is rather preferred over the sigmoid function?



BACKPROPAGATION

The standard approach is to use maximum likelihood estimation, by minimizing NLL:

$$\mathcal{L}(\theta) = -\sum_{n=1}^{N} \log p(\mathbf{y_n}|\mathbf{x_n};\theta)$$

It is also common to add a regularizer and minimizes the PNLL:

$$\mathcal{L}(\theta) = -\sum_{n=1}^{N} \log p(\mathbf{y_n}|\mathbf{x_n};\theta) - \lambda \log p(\theta)$$

To optimize the objective function, we need to compute the gradient via Backpropagation

BACKPROPAGATION

To better understand the Backpropagation, let's consider a mapping of the form $o = f(x)$, where $x \in \mathbb{R}^n$ and $o \in \mathbb{R}^m$. We assume that f is defined as a composition of functions:

$$f = f_4 \circ f_3 \circ f_2 \circ f_1$$

The intermediate steps needed to compute $o = f(x)$ are $x_2 = f_1(x)$, $x_3 = f_2(x_2)$, $x_4 = f_3(x_3)$, and $o = f_4(x_4)$. We can compute the Jacobian $J_f(x) = \frac{\partial o}{\partial x} \in \mathbb{R}^{m \times n}$ using the chain rule:

$$\frac{\partial o}{\partial x} = \frac{\partial o}{\partial x_4}\frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial x} = \frac{\partial f_4(x_4)}{\partial x_4}\frac{\partial f_3(x_3)}{\partial x_3}\frac{\partial f_2(x_2)}{\partial x_2}\frac{\partial f_1(x)}{\partial x}$$

BACKPROPAGATION

Example

Given the following loss function $\mathcal{L}(\theta) = \frac{1}{2}||y - W_2\psi(W_1x)||_2^2$, represent the forward model and the gradient w.r.t the parameters.

Forward step:

$$\mathcal{L} = \boldsymbol{f}_4 \circ \boldsymbol{f}_3 \circ \boldsymbol{f}_2 \circ \boldsymbol{f}_1$$
$$\boldsymbol{x}_2 = \boldsymbol{f}_1(\boldsymbol{x}, \boldsymbol{\theta}_1) = \mathbf{W}_1\boldsymbol{x}$$
$$\boldsymbol{x}_3 = \boldsymbol{f}_2(\boldsymbol{x}_2, \emptyset) = \varphi(\boldsymbol{x}_2)$$
$$\boldsymbol{x}_4 = \boldsymbol{f}_3(\boldsymbol{x}_3, \boldsymbol{\theta}_3) = \mathbf{W}_2\boldsymbol{x}_3$$
$$\mathcal{L} = \boldsymbol{f}_4(\boldsymbol{x}_4, \boldsymbol{y}) = \frac{1}{2}||\boldsymbol{x}_4 - \boldsymbol{y}||^2$$

Backward step:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_3} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_4}\frac{\partial \boldsymbol{x}_4}{\partial \boldsymbol{\theta}_3}$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_2} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_4}\frac{\partial \boldsymbol{x}_4}{\partial \boldsymbol{x}_3}\frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{\theta}_2}$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_1} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_4}\frac{\partial \boldsymbol{x}_4}{\partial \boldsymbol{x}_3}\frac{\partial \boldsymbol{x}_3}{\partial \boldsymbol{x}_2}\frac{\partial \boldsymbol{x}_2}{\partial \boldsymbol{\theta}_1}$$

BACKPROPAGATION

Algorithm 7: Backpropagation for an MLP with K layers

```
// Forward pass
x_1 := x
for k = 1 : K do
    x_{k+1} = f_k(x_k, θ_k)
// Backward pass
u_{K+1} := 1
for k = K : 1 do
    g_k := u_{k+1}^T ∂f_k(x_k, θ_k)/∂θ_k
    u_k^T := u_{k+1}^T ∂f_k(x_k, θ_k)/∂x_k
// Output
Return L = x_{K+1}, ∇_x L = u_1, {∇_{θ_k} L = g_k : k = 1 : K}
```

$$\boldsymbol{x}_1 := \boldsymbol{x}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)$$

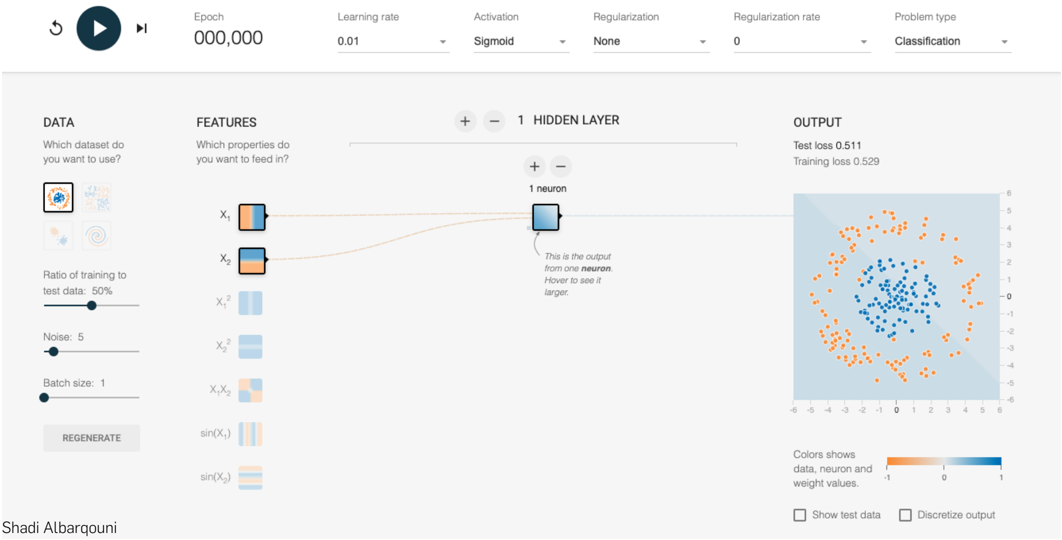
$$\boldsymbol{u}_{K+1} := 1$$

$$\boldsymbol{g}_k := \boldsymbol{u}_{k+1}^\top \frac{\partial \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)}{\partial \boldsymbol{\theta}_k}$$

$$\boldsymbol{u}_k^\top := \boldsymbol{u}_{k+1}^\top \frac{\partial \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{\theta}_k)}{\partial \boldsymbol{x}_k}$$

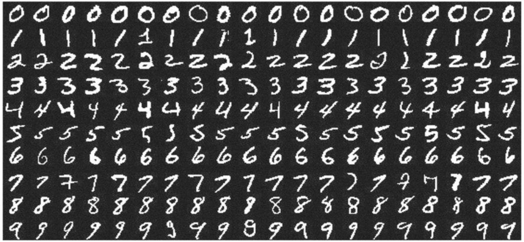
$$\text{Return } \mathcal{L} = \boldsymbol{x}_{K+1},\ \nabla_{\boldsymbol{x}}\mathcal{L} = \boldsymbol{u}_1,\ \{\nabla_{\boldsymbol{\theta}_k}\mathcal{L} = \boldsymbol{g}_k : k = 1 : K\}$$

DEMO


Epoch
000,000
Learning rate
0.01
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 5
Batch size: 1
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.511
Training loss 0.529
Colors shows data, neuron and weight values.
Show test data
Discretize output

MLP FOR IMAGE CLASSIFICATION - MNIST

MNIST Dataset

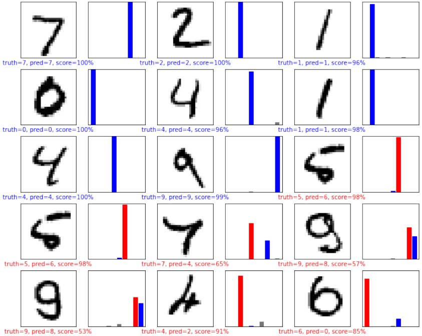


The key idea is to either flatten the image into a fixed-dimensional vector or exctract handcrafted features.

Does randomly shuffling the pixels affect the output of the MLP model – presuming the same shuffle applies for all inputs?

MLP FOR IMAGE CLASSIFICATION - MNIST

Recognition Model

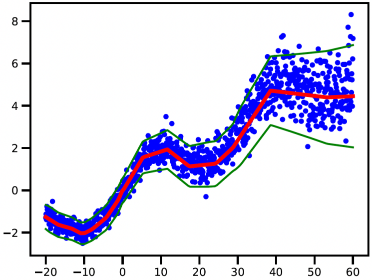


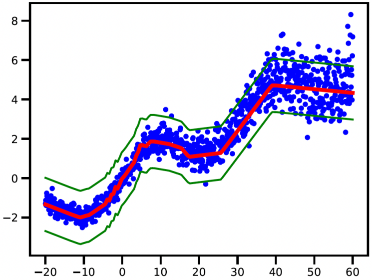
How many paramters in the MLP Recognition model?

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape
=================================================================
flatten (Flatten)            (None, 784)
_________________________________________________________________
dense (Dense)                (None, 128)
_________________________________________________________________
dense_1 (Dense)              (None, 128)
_________________________________________________________________
dense_2 (Dense)              (None, 10)
=================================================================
```

MLP FOR HETREOSKEDASTIC REGRESSION

In linear regression: $\mathcal{N}(y|\mathbf{w}_\mu^T\mathbf{x} + b, \sigma_+(\mathbf{w}_\sigma^T\mathbf{x}))$

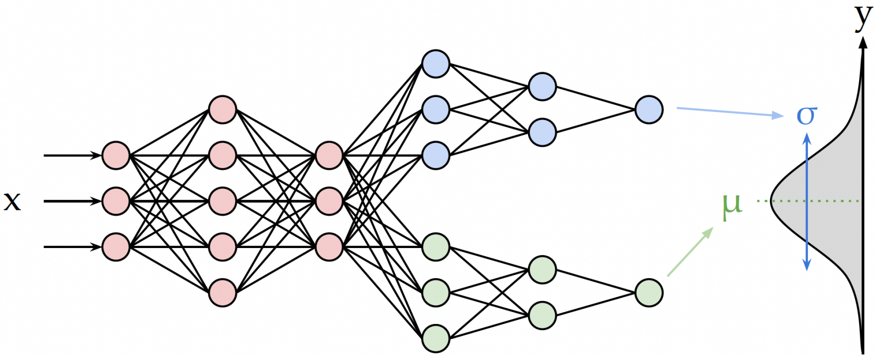




How can we model the hetreoskedastic regression in MLPs?

MLP FOR HETREOSKEDASTIC REGRESSION

In linear regression: $\mathcal{N}(y|\mathbf{w}_\mu^T\mathbf{x} + b, \sigma_+(\mathbf{w}_\sigma^T\mathbf{x}))$

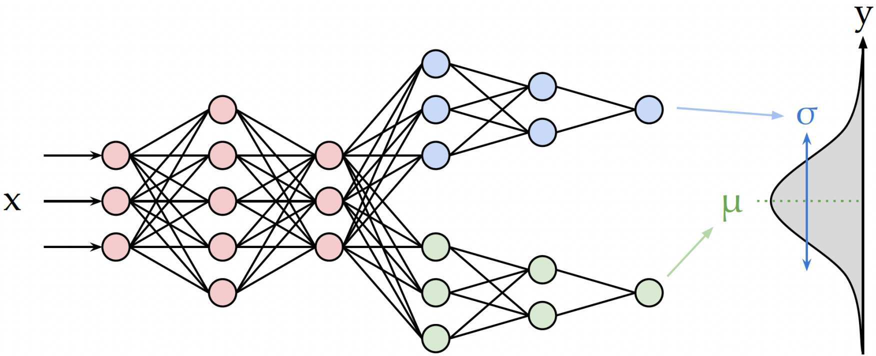


Source: **https://brendanhasz.github.io/2019/07/23/bayesian-density-net.html**

MLP FOR HETREOSKEDASTIC REGRESSION

In linear regression: $\mathcal{N}(y|\mathbf{w}_\mu^T\mathbf{x} + b, \sigma_+(\mathbf{w}_\sigma^T\mathbf{x}))$

In MLP regression: $\mathcal{N}(y|\mathbf{w}_\mu^T f(\mathbf{x};\theta_{shared}), \sigma_+(\mathbf{w}_\sigma^T f(\mathbf{x};\theta_{shared})))$

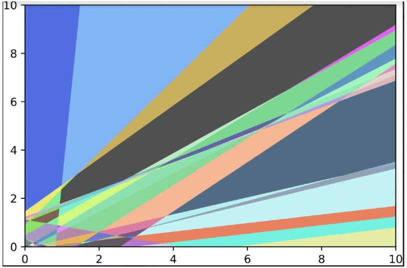


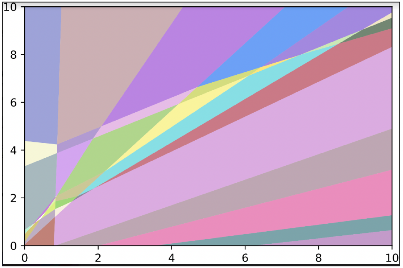
Source: **https://brendanhasz.github.io/2019/07/23/bayesian-density-net.html**

TECHNICAL ISSUES

THE IMPORTANCE OF DEPTH

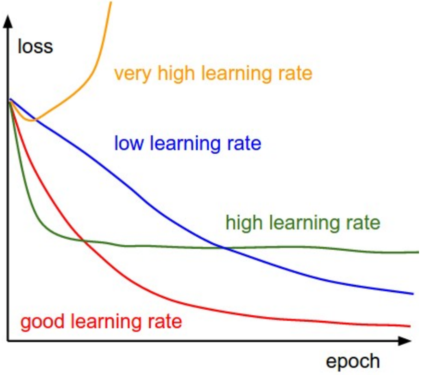
One can show that an MLP with one hidden layer is a universal function approximator, meaning it can model any suitably smooth function, given enough hidden units, to any desired level of accuracy. However, various arguments, both experimental and theoretical have shown that deep networks work better than shallow ones. Take the XOR challenge as an example.



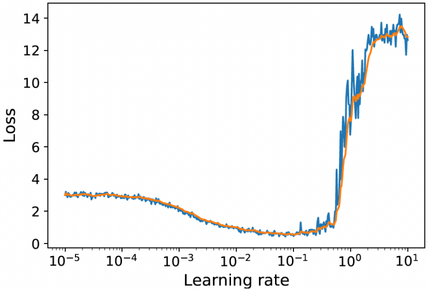


CHOOSING THE LEARNING RATE

We need to be careful in how we choose the learning rate in order to achieve convergence



Source: **https://cs231n.github.io**



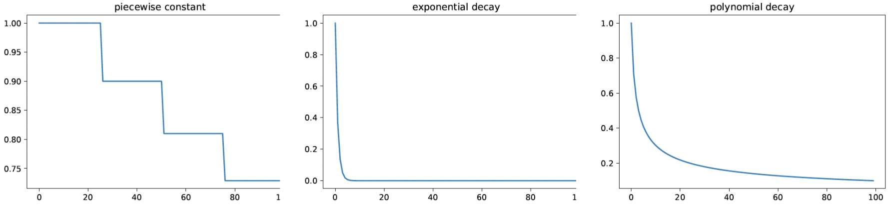
CHOOSING THE LEARNING RATE

Rather than choosing a single constant learning rate, we can use a learning rate schedule, in which we adjust the step size over time.

piecewise constant: $\eta_t = \eta_i$ if $t_i \leq t \leq t_{i+1}$

exponential decay: $\eta_t = \eta_0 e^{-\lambda t}$

polynomial decay: $\eta_t = \eta_0 (\beta t + 1)^{-\alpha}$



WEIGHT INITIALIZATION

It has been shown that sampling parameters from a standard normal with fixed variance can result in exploding activations or gradients.

Xavier initialization: $\sigma^2 = \frac{2}{n_{in}+n_{out}} \rightarrow$ linear, tanh, logistic, and softmax.

LeCun initialization: $\sigma^2 = \frac{1}{n_{in}} \rightarrow$ SELU

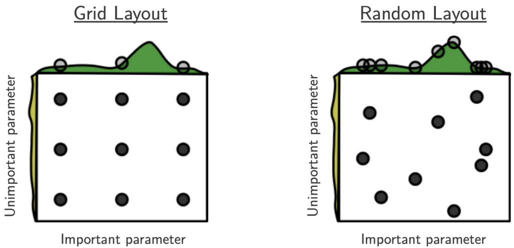
He initialization: $\sigma^2 = \frac{2}{n_{in}} \rightarrow$ ReLU and its variants.

where n_{in} is the fan-in of a unit (number of incoming connections), and n_{out} is the fan-out of a unit (number of outgoing connections).

CHOOSING HYPER-PRAMETERS

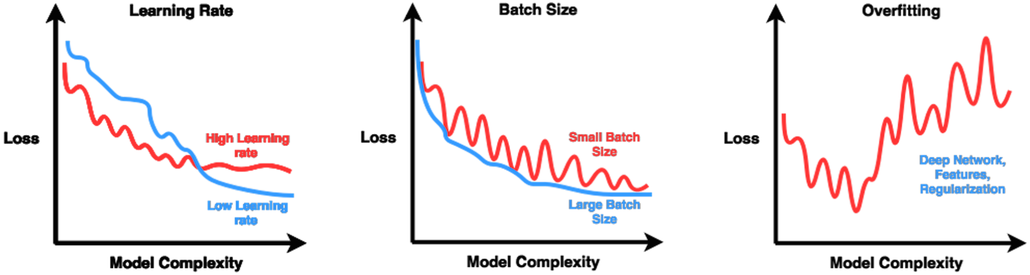
The recipe:

(1) Check the initial loss
(2) Overfit a small subset
(3) Find the learning rate that lower the loss
(4) Coarse grid with a few epochs
(5) Refine grid with longer epcohs
(6) Observe the loss and accuracy

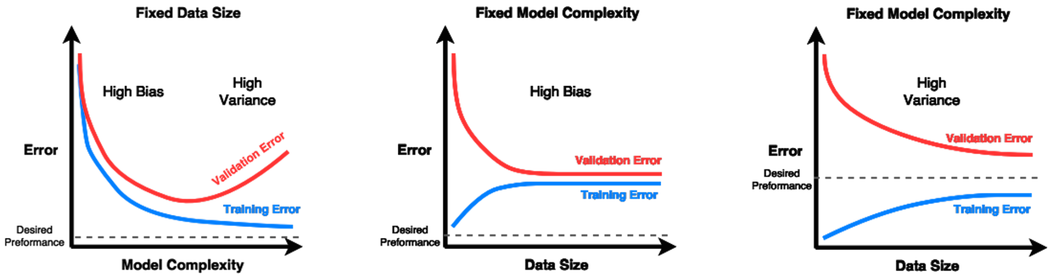


Source: **https://cs231n.github.io/neural-networks-3/**

CHOOSING HYPER-PRAMETERS



DEBUGGING THE MODEL



Questions