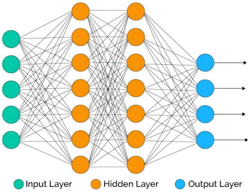
Machine learning for particle physicists

III. How to train better networks

Anja Butter

26th Vietnam School of Physics

Recap - Neural networks basics



DNN:	$\tilde{\boldsymbol{y}} = \sigma_n(\boldsymbol{W}_n \cdot \sigma_{n-1}(\boldsymbol{W}_{n-1} \cdot \ldots \cdot \sigma_1(\boldsymbol{W}_1 \boldsymbol{X})))$
Backpropagation:	$\nabla_{\boldsymbol{w}_i} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \sigma_n} \frac{\partial \sigma_n}{\partial lin_n} \frac{\partial lin_n}{\partial \sigma_{n-1}} \ldots \frac{\partial lin_i}{\partial w_i}$
Mini-batch gradient descent:	weight update based on batch of data
Overfitting:	network learns noise
	$\rightarrow$ control: validation + test dataset

Plan for today

How to train better networks

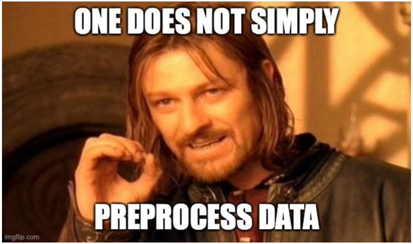
Practical tips to avoid frustration :)

1. Data preprocessing
2. Network initialization
3. Optimizing the training procedure
4. Regularization
5. Hyperparameter tuning

Data Preprocessing

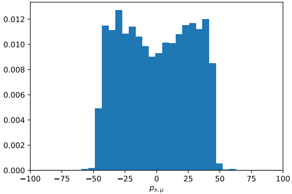
Why preporcessing?

- input features with different scales
  eg. jet = (charge, $n_{particles}, p_T, M, \eta, \phi$)
- large value with small spread
  eg. $pp \rightarrow Z \rightarrow ll, m_{ll} \in$ [80 GeV − 100 GeV]
- weights usually initialized to be sensitive in range [-1,+1]
- classification output in range [0,1]
- training more efficient/stable if features are also in range [-1, +1]

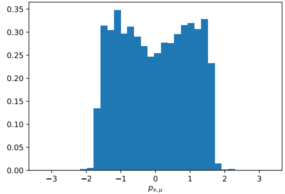


Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Rule of thumb: rescale to $\mu = 0, \sigma = 1$

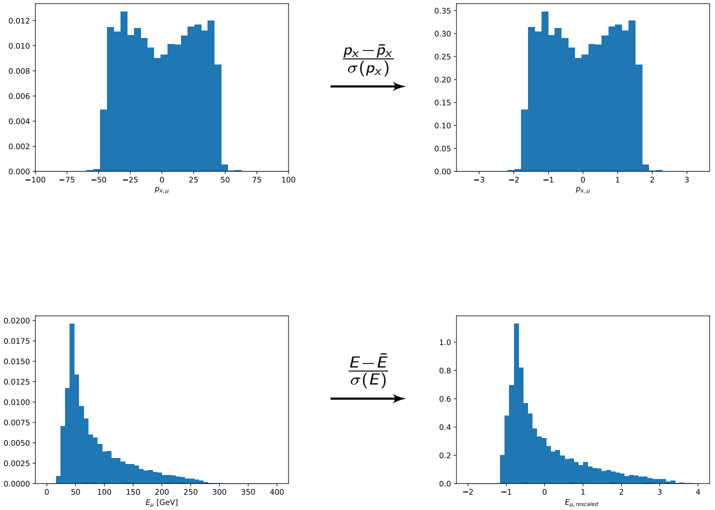


$$\xrightarrow{\frac{p_x - \bar{p}_x}{\sigma(p_x)}}$$



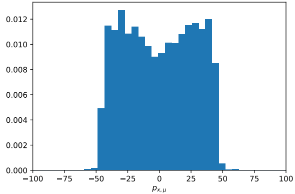
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Rule of thumb: rescale to $\mu = 0, \sigma = 1$

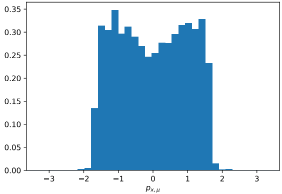


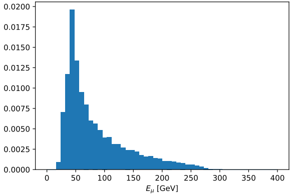
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Rule of thumb: rescale to $\mu = 0, \sigma = 1$



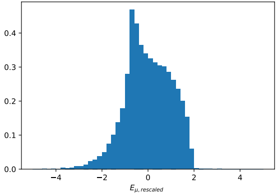
$$\xrightarrow{\frac{p_x - \bar{p}_x}{\sigma(p_x)}}$$





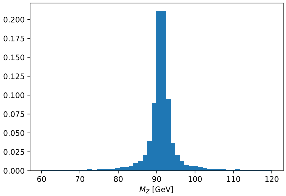
$$\xrightarrow{\frac{E' - \bar{E}'}{\sigma(E')}}$$

$E' = \log(E - 20)$

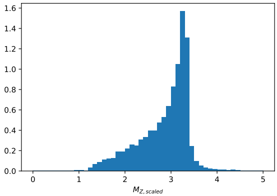


Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Exception: Correlated observables

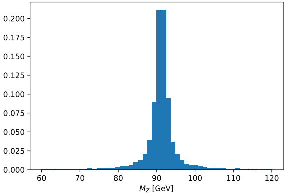


$$\xrightarrow{\frac{p_{i,\mu} - \bar{p}_{i,\mu}}{\sigma(p_{i,\mu})}}$$

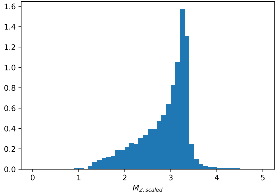


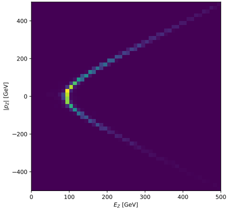
Rescaling

Example: $pp \to Z \to \mu^+\mu^-$
Exception: Correlated observables

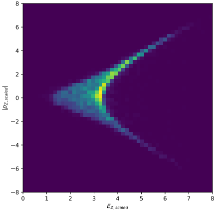


$$\xrightarrow{\frac{p_{i,\mu}-\bar{p}_{i,\mu}}{\sigma(p_{i,\mu})}}$$





$$\xrightarrow{\frac{p_{i,\mu}-\bar{p}_{i,\mu}}{\sigma(p_{i,\mu})}}$$



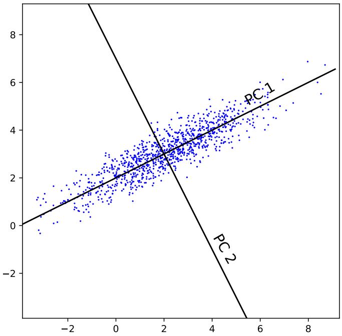
$\Rightarrow$ Use same scale for $p_{i,\mu}$

PCA

Principal component analysis

- directions maximizing variance
- eigenvector of covariance matrix
- $\mathrm{cov}(\boldsymbol{X}) = \boldsymbol{X}^T \boldsymbol{X}$

+ facilitates training
+ useful for interpretation
+ can reduce data dimension



2. Network initialization

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$?

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$?

   symmetric initialization $\Rightarrow$ symmetric updates $\Rightarrow$ identical weights $\lightning$

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1?$ ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)?$

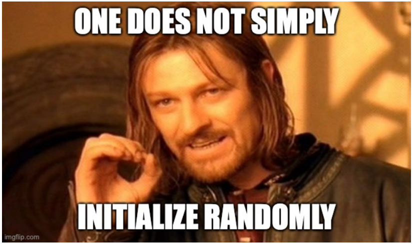
Check for single neuron $y = w_i x_i$ with w_i, x_i independent:

$$
\begin{aligned}
< y^2 > &= \sum_i < w_i^2 x_i^2 > \\
&= \sum_i < w_i >^2 < x_i^2 > + < x_i >^2 < w_i^2 > + < w_i^2 >< x_i^2 > \\
&= \sum_i < w_i^2 >< x_i^2 > \quad \leftarrow < w_i > = < x_i > = 0 \\
&= n_{incoming} < w_i^2 >< x_i^2 > \text{ diverges!}
\end{aligned}
$$

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1?$ ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)?$ ↯



$\rightarrow < w_i^2 > = \frac{1}{n_{incoming}}$ to preserve variance through network

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$? ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$? ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
    - caveat 1: Same argument for backpropagation $\rightarrow$ average $(n_{in} + n_{out})/2$
    - caveat 2: only for $\approx$ linear activation function eg. tanh

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1?$ ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)?$ ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
4. ReLU $\rightarrow$ 50% of outputs $= 0 \rightarrow$ additional factor 2
   $\Rightarrow$ He initialization $\sigma = \sqrt{2/n_{in}}$

Network initialization

We know how to update weights. But how do we start?

1. $w_i = 1$? ↯
2. $w_i \sim \mathcal{N}(\mu = 0, \sigma = 1)$? ↯
3. Xavier/Glorot initialization $w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{2/(n_{in} + n_{out})}\right)$
4. ReLU $\rightarrow$ 50% of outputs $= 0 \rightarrow$ additional factor 2
   $\Rightarrow$ He initialization $\sigma = \sqrt{2/n_{in}}$
5. Glorot & He initialization also available for uniform distributions

Pretraining

For some tasks we can use pretrained networks
$\rightarrow$ trained on large dataset to extract image features

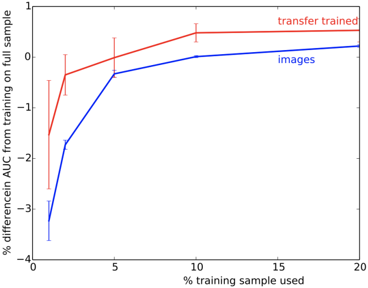
Google's InceptionResNetV2 to identify anomalies in QCD jet images
Best image class to identify QCD jet images:

Pretraining

For some tasks we can use pretrained networks
$\rightarrow$ trained on large dataset to extract image features

Google's InceptionResNetV2 to identify anomalies in QCD jet images
Best image class to identify QCD jet images:

Lisa Benato



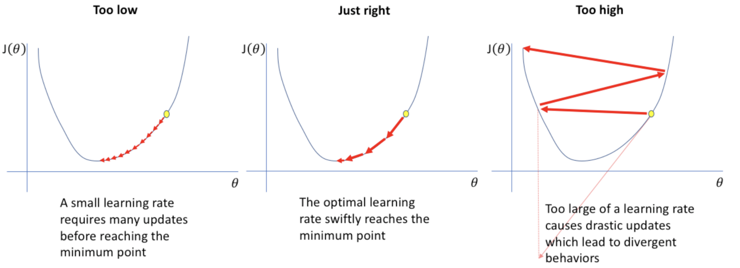
Ice cream classification :)

3. Optimizing the training procedure

Optimizing the training procedure

Reminder

Convergence depends on learning rate



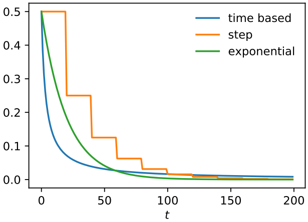
https://www.jeremyjordan.me/nn-learning-rate/

$\rightarrow$ Experiment with different orders of magnitude eg. $10^{-1} \dots 10^{-6}$

Learn rate decay

Reduce learning rate over time to improve convergence

Time-Based Decay	$l(t) = \frac{l_0}{1 + k * t}$	
Step Decay	$l(t) = l_0 * \lambda^{int(t/\tau)}$	with $0 < \lambda < 1$
Exponential Decay	$l(t) = l_0 * e^{-t/\tau}$	

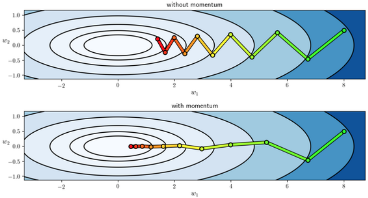


Momentum

Problem: One dimension much steeper than the other

$$
\begin{aligned}
&\text{gradient descent} \qquad && \boldsymbol{W}_t \to \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \nabla_{\boldsymbol{W}_t} \mathcal{L} \\
&\text{GD + momentum} \qquad && \boldsymbol{W}_t \to \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha v_{dw} \\
& && v_{dw} = \beta v_{dw} + (1-\beta) \nabla_{\boldsymbol{W}_t} \mathcal{L}
\end{aligned}
$$

Intuition: ball picks up momentum



jermwatt.github.io/machine_learning_refined

enforces dimensions where gradient points in same direction
+ reduces oscillation

Adagra/RMSprop

Adapt updates to individual parameters

$$\boldsymbol{W}_t \to \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \frac{1}{\sqrt{G + \epsilon}} \nabla_{\boldsymbol{W}_t} \mathcal{L}$$

$\rightarrow$ Different learning rate for each parameter

Adagrad: $$G_{ii,t} = \sum_{t'=0}^{t} \mathrm{d}w_{i,t'}^2$$

sum over vector of all past gradients

$\rightarrow$ monotonically decreasing learning rate

RMSprop: $$G_{ii,t} = \beta G_{ii,t-1} + (1-\beta)\mathrm{d}w_{i,t}^2$$

$\rightarrow \beta = 0.9 \rightarrow$ decaying average

Adam

Adaptive moment estimation

Standard go to option, stable & fast
Combines first moment (momentum) and second moment (RMSprop)

$$\boldsymbol{W}_t \to \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \alpha \frac{1}{\sqrt{G + \epsilon}} v_{dw}$$

$$v_{dw,t} = \frac{1}{1-\beta_1}\left(\beta_1 v_{dw,t-1} + (1-\beta_1) dw_{i,t}\right)$$

$$G_{ii,t} = \frac{1}{1-\beta_2}\left(\beta_2 G_{ii,t-1} + (1-\beta_2) dw_{i,t}^2\right)$$

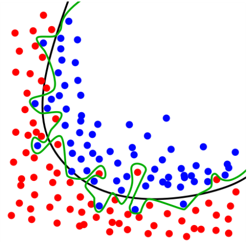
Others worth exploring!
Might fit your problem better?

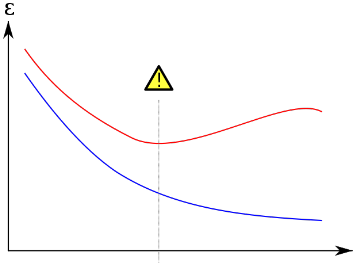
- Nesterov accelerated gradient
- Adadelta
- AMSGrad

4. Regularization

Reminder: Overtraining

Overtraining: networks picks up irrelevant features





Control with validation/test data

We refer to this regularization technique as early stopping

not always applicable → consider alternatives

Regularization

Modify network

- Dropout
- Batch normalization

Modify loss

- l_1 regularization
- l_2 regularization
- gradient penalty

Modifying the loss

The network is constrained by punishing large weight values

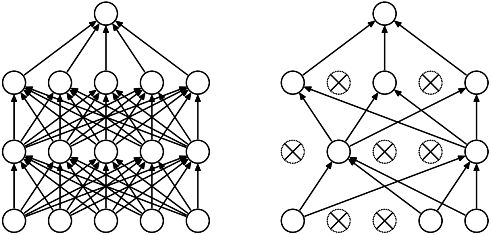
$$\mathcal{L} = \mathcal{L}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) + \alpha\Omega(\boldsymbol{W})$$

l_1	$\Omega(\boldsymbol{W}) = \|\|\boldsymbol{W}\|\|_1 = \sum_{ij} \|W_{ij}\|$
l_2	$\Omega(\boldsymbol{W}) = \|\|\boldsymbol{W}\|\|_2^2 = \sum_{ij} W_{ij}^2$
gradient penalty	$\Omega(\boldsymbol{W}) \sim \|\|\nabla_{\boldsymbol{x}} \tilde{\boldsymbol{y}}\|\|_2^2$

Dropout

Randomly switching off nodes during training

Intuition: Train many different models, then average for evaluation



(a) Standard Neural Net (b) After applying dropout.

2014, N. Srivastava et. al

Batch normalization

Idea: fix mean and variance of layer output

$$\boldsymbol{x} \to \boldsymbol{x}_{norm} = \frac{\boldsymbol{x} - \bar{\boldsymbol{x}}}{\sqrt{\sigma(\boldsymbol{x}) + \epsilon}}$$

$$\boldsymbol{y} = \gamma \boldsymbol{x}' + \beta$$

trainable parameters γ, β

During training: normalization per batch
For inference: normalization from full dataset

Why it works is subject of current research!
Smoothness of optimization landscape? Length-Direction decoupling?

5 Hyperparameter tuning

Hyperparameter tuning

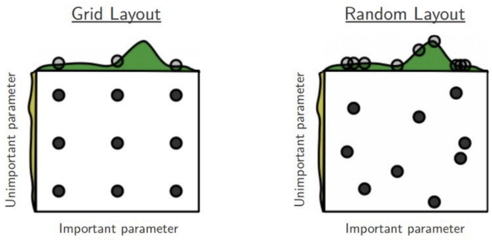
How can we find the best settings for the training?
Problem: We can not compute a gradient!

Hyperparameter tuning

How can we find the best settings for the training?
Problem: We can not compute a gradient!

1. by hand $\rightarrow$ underrated, helps to build experience
2. Grid search
3. Random (blind)
4. Bayesian optimization (educated guess, advanced)

Advantage of random vs grid search



Advantages: easy to code, run parallel
Disadvantage: no use of information from previous iterations, curse of dimensionality

All the things you can do to your ML setup

1. Data preprocessing
   - Rescaling, PCA
2. Network initialization
   - Glorot/HE, Normal/uniform
3. Optimizing the training procedure
   - Learning rate scheduling, momentum, Adagrad, Adam
4. Regularization
   - Via early stopping, additional loss, dropout, or Batchnorm
5. Hyperparameter tuning
   - get a feeling for the network, random search, Bayesian optimization

Ready to try it out?

$\rightarrow$ colab
$\rightarrow$ gitHub
$\rightarrow$ dhrou
$\rightarrow$ HEPMLtutorials
$\rightarrow$ HEPML_HandsOn_NN.ipynb

Big thank you to David Rousseau for sharing this tutorial!

Corrections

```
D = Model(inputs=[inputs], outputs=[Dx])
class_weight = {
0: class_weights[0],
1: class_weights[1],
}
D.fit(
X_train,
y_train.values,
epochs=10,
verbose=0,
class_weight=class_weight
)
```