Deep learning

Optimization and Regularization in deep networks

Hamid Beigy

Sharif University of Technology

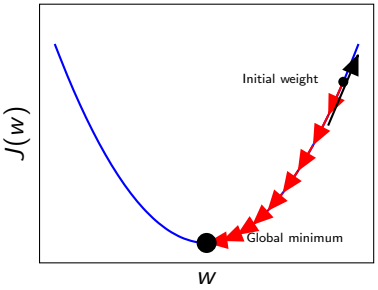
October 31, 2025

Table of contents

Introduction

Gradient based learning

1. Gradient based learning is by far the most popular optimization strategy, used in machine learning and deep learning at the moment.
2. Cost (error) is a function of the weights (parameters).
3. We want to reduce/minimize the error.
4. Gradient descent: move towards the error minimum.
5. Compute gradient, which implies get direction to the error minimum.
6. Adjust weights towards direction of lower error.



Optimization methods

1. Based on the gradient information in optimization, optimization methods can be divided into three categories (Sun et al. 2019):

   first-order optimization methods such as stochastic gradient methods,

   high-order optimization methods such as Newton's method, and

   heuristic derivative-free optimization methods coordinate descent methods.

2. First-order optimization methods, has been widely used in recent years.
3. High-order methods converge at a faster speed than first-order methods. They have difficulty in the operation and storage of the inverse of the Hessian matrix.
4. Derivative-free optimization methods are used when the derivative of the objective function may not exist or be difficult to calculate. Their ideas:
   - adopting a heuristic search based on empirical rules, and
   - fitting the objective function with samples.
5. Derivative- free optimization methods can also work in conjunction with gradient-based methods.

Examples of optimization in machine learning

1. Almost all machine learning algorithms can be formulated as an optimization problem to find the extremum of an objective function.
2. In supervised learning, we have a training set

$$S = \{(x_1, t_1), (x_2, t_2), \ldots, (x_m, t_m)\}$$

3. For example, in regression task the objective function for a Perceptron with sigmoid unit is defined as

$$\arg\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - t_i)^2$$

4. For example, in classification task the objective function for a Perceptron with sigmoid unit is defined as

$$\arg\min_\theta \sum_{i=1}^{m} \left[-t^i \ln h(x^i) - (1 - t^i) \ln(1 - h(x^i)) \right]$$

First-order optimization methods

Gradient descent

1. Gradient descent (GD), computes the gradient of the cost function w.r.t. to the parameters θ.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta)$$

2. We need to calculate the gradients for the whole dataset to perform just one update.
3. GD can be very slow and is intractable for datasets that don't fit in memory.
4. GD also doesn't allow us to update our model online, i.e. with new examples on-the-fly.
5. GD method is simple to implement.
6. The solution is global optimal when the objective function is convex.
7. It often converges at a slower speed if the variable is closer to the optimal solution, and more careful iterations need to be performed.

Linear regression (A simple demo)

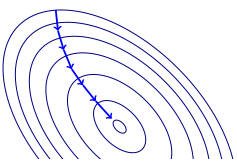
Consider a linear regression problem with MSE loss.

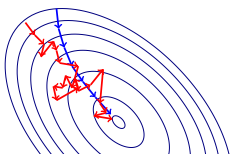
This demo was taken from we this page.

1. GD has high computational complexity in each iteration for large-scale data and does not allow online update.
2. One solution is using stochastic gradient descent (SGD).
3. The ides of SGD is using one sample randomly to update the gradient per iteration, instead of directly calculating the exact value of the gradient.
4. SGD performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

5. It is therefore usually much faster and can also be used to learn online.
6. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily[1].





[1] Figures are taken from slides of Trivedi & Kondor

Mini-batch gradient descent

1. Mini-batch gradient descent (MGD) finally takes the best of both worlds and performs an update for every mini-batch of K training examples.

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+K)}; y^{(i:i+K)})$$

2. This method reduces the variance of the parameter updates, which can lead to more stable convergence and helps to improve the optimization speed.
3. It can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient (mini-batch is very efficient).
4. Common mini-batch sizes range between 50 and 256, but can vary for different applications.

Mini-batch gradient descent (Challenges)

Mini-batch gradient descent does not guarantee good convergence and offers a few challenges that need to be addressed.

1. Choosing a proper learning rate can be difficult.
2. Choosing the parameters (schedules and thresholds) of learning rate schedules is difficult.
3. Are we using the same learning rate for all parameters?
4. How to avoid from getting trapped in suboptimal local minima.

Momentum

1. Momentum is a method that helps accelerate GD in the relevant direction and dampens oscillations (Qian 1999).
2. Its concept is derived from the mechanics of physics, which simulates the inertia of objects.
3. The idea of applying momentum is to preserve the influence of the previous update direction on the next iteration to a certain degree.
4. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta_{t+1} = \theta_t - v_t$$

5. In GD, the speed update is $-\eta \nabla_\theta J(\theta)$ each time.
6. Using the momentum algorithm, the amount of the update v is not just the amount of GD calculated by $-\eta \nabla_\theta J(\theta)$. It also takes into account the friction factor, which is represented as the previous update v_{t-1} multiplied by a momentum factor γ.

Nesterov accelerated gradient

1. Nesterov Accelerated Gradient Descent (NAG) makes further improvement over the momentum method (Nesterov 1983).
2. A ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.
3. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
4. NAG is a way to give our momentum term this kind of prescience.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_t - \gamma v_{t-1})$$
$$\theta_{t+1} = \theta_t - v_t$$

5. Value of $\theta_t - \gamma v_{t-1}$ gives an approximation of the next position of the parameters.
6. The improvement of NAG over momentum is reflected in updating the gradient of the future position instead of the current position.
7. NAG improves convergence rate
8. It is hard to determine the learning rate and usually it oscillates in the near of the optimal point. The learning rate is updated using

$$\eta_t = \frac{\eta_0}{1 + d \times t} \quad d \in [0, 1]$$

Adagrad

1. Manually regulated learning rate greatly influences the effect of the SGD method.
2. It is a tricky problem for setting an appropriate value of the learning rate.
3. AdaGrad adjusts the learning rate dynamically based on the historical gradient in some previous iterations (Duchi, Hazan, and Singer 2011).
   - Smaller updates for parameters associated with frequently occurring features.
   - Larger updates for parameters associated with infrequent features.
4. This algorithm well-suited for dealing with sparse data.
5. Adagrad uses a different learning rate for every parameter θ_i at every time step t.
6. Adagrad updates the parameters in the following manner.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_\theta J(\theta_{t,i})$$

   where $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element (i, i) is the sum of the squares of the gradients w.r.t. θ_i up to time t. ϵ is a smoothing term that avoids division by zero.

7. Without the square root operation, the algorithm performs much worse.
8. AdaGrad is good when the objective is convex.

Adam

1. Adaptive moment estimation (Adam) computes adaptive learning rates for each parameter (Kingma and J. Ba 2015).
2. Adam behaves like a heavy ball with friction, which prefers flat minima in the error surface.
3. Adam computes the decaying averages of past and past squared gradients m_t and v_t, respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2$$

where m_t and v_t are estimates of the first and the second moments of the gradients, respectively,

Adam (cont.)

1. Initially m_t and v_t are set to 0.
2. m_t and v_t are biased towards zero, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).
3. Bias-corrected m_t and v_t are

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

Then Adam updates parameters as (same as Adadelta and RMSprop)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

AMSGrad

1. Adaptive learning rate methods have become the norm in training neural networks but in some applications they fail to converge to an optimal solution and are outperformed by GD with momentum.
2. Some reasons:
   - The first one is exponential moving average of past squared gradients.
   - The second one is some mini-batches provide large and informative gradients but occur rarely.
3. AMSGrad that uses the maximum of past squared gradients v_t rather than the exponential average to update the parameters (Reddi, Kale, and Kumar 2018).
4. In AMSGrad, v_t is defined the same as in Adam.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta J(\theta_t))^2$$

5. Instead of using v_t directly, we now employ the previous v_{t-1} if it is larger than the current one:

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

AMSGrad (cont.)

1. This way, AMSGrad results in a non-increasing step size, which avoids the problems suffered by Adam.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta J(\theta_t)$$

$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} m_t$$

2. The authors observe improved performance compared to Adam on small datasets and on CIFAR-10.
3. Other experiments show similar or worse performance than Adam.

Other optimization algorithms

1. RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates.
2. AdaMax changed parameter v_t of Adam and generalized it to L_p norm(Kingma and J. Ba 2015).
3. Nadam (Nesterov-accelerated Adaptive Moment Estimation)(Dozat 2016) combines Adam and NAG.
4. For more information please read this paper(Ruder 2016; Sun et al. 2019).
5. Which optimizer to use?

Comparison of some optimization algorithms

Consider contours of a loss surface and time evolution of different optimization algorithms.

Notice the overshooting behavior of momentum-based methods. This demo was taken from we this page.

Comparison of some optimization algorithms

Consider a visualization of a saddle point in the optimization landscape.

Notice the hard convergence behavior of SGD. This demo was taken from we this page.

Model selection

1. Considering regression problem, in which the training set is

$$S = \{(x_1, t_1), (x_2, t_2), \ldots, (x_m, t_m)\}, t_k \in \mathbb{R}.$$

where

$$t_k = f(x_k) + \epsilon \qquad \forall k = 1, 2, \ldots, m$$

$f(x_k) \in \mathbb{R}$ is the unknown function and ϵ is the random noise.

2. The goal is to approximate the $f(x)$ by a function $g(x)$.
3. The empirical error on the training set S is measured using cost function

$$J(g) = \frac{1}{2} \sum_{i=1}^{m} (t_i - g(x_i))^2$$

4. The aim is to find $g(.)$ that minimizes the empirical error.
5. We assume that a hypothesis class for $g(.)$ with a small set of parameters.
6. Let $g(x)g = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$.

1. Define the following vectors and Matrix
   - Data matrix

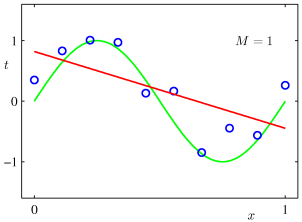
$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & & & \vdots & \\ 1 & x_{N1} & x_{N2} & \dots & x_{DD} \end{bmatrix}$$

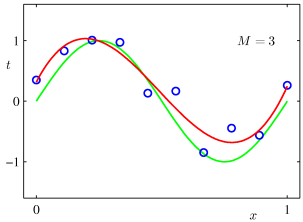
   - The k^{th} input vector: $X_k = (1, x_{k1}, x_{k2}, \dots, x_{kD})^T$
   - The weight vector: $W = (w_0, w_1, w_2, \dots, w_D)^T$
   - The target vector: $t = (t_1, t_2, t_3, \dots, t_N)^T$
2. The empirical error equals to: $J(g) = \frac{1}{2} \sum_{k=1}^{m} \left(t_k - W^\top X_k\right)^2$.
3. The gradient of $J(g)$ equals to $\nabla_W J(g) = \sum_{k=1}^{m} t_k X_k^\top - W^\top \sum_{k=1}^{m} X_k X_k^\top = 0$
4. Solving for W, we obtain $W^* = \left(X^\top X\right)^{-1} X^\top t$

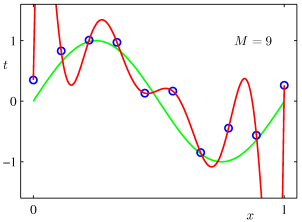
1. If the linear model is too simple, the model can be a polynomial (a more complex hypothesis set)

$$g(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M.$$

2. M is the order of the polynomial.
3. Choosing the right value of M is called model selection.







4. For $M = 1$, we have a too general model
5. For $M = 9$, we have a too specific model

Training vs Testing error

1. Given a new data point x, we would like to understand the expected prediction error

$$\mathbb{E}\big[(t - g(x))^2\big]$$

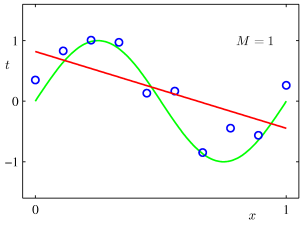
2. Assume that x generated by the same process as the training set. We decompose $\mathbb{E}\big[(t - g(x))^2\big]$ into bias, variance, and noise.

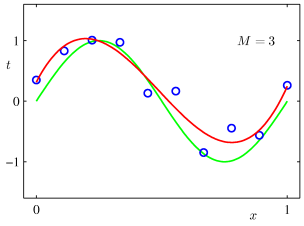
$$\mathbb{E}\big[(t - g(x))^2\big] = Bias^2(g(x)) + Var(g(x)) + \sigma^2$$

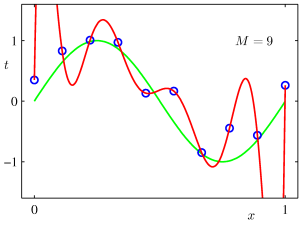
3. The first term describes the average error of $g(x)$.
4. The second term quantifies how much $g(x)$ deviates from one training set S to another one. This depends on both the estimator and the training set.This term is consequence of over-fitting.
5. The last term is the variance of the added noise. This error cannot be removed no matter what estimator we use. Note that the variance of the noise can not be minimized.

Training vs Testing error

1. In regression, as M increases, a small changes in the data sets causes a greater change in the fitted function; thus variance increases.
2. The goal is to minimize the expected loss. There is a trade-off between bias and variance.
   - Very flexible models have low bias and high variance.
   - Relative rigid models have high bias and low variance.
3. The model with optimal predictive capability is one that leads to the best balance between bias and variance.
4. If there is bias, there is under-fitting. why?
5. If there is variance, there is over-fitting. why?

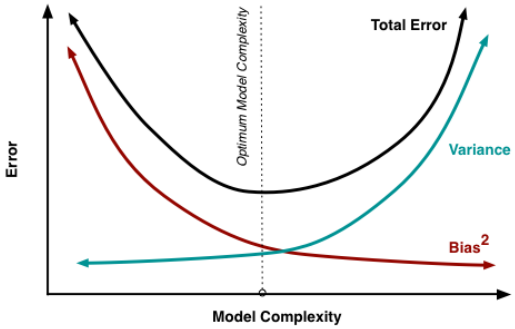






Bias-Variance trade-off

1. Consider bias-variance of a model.



2. The problem is how to balance between bias and variance.
3. Some solutions
   - Trial and error using validation dataset
   - Regularization
   - ⋮

Regularization

Regularization

1. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
2. In this manner, regularization is used to improve the generalization of the model. Other intuitions:
   - Regularization tends to increase the estimator bias while reducing the estimator variance.
   - Regularization can be seen as a way to prevent overfitting.
   - A common problem is in picking the model size and complexity. It may be appropriate to simply choose a large model that is regularized appropriately.
3. Regularization can also be studied from view point of statistical learning theory

Types of regularization

1. Regularizations may take on many different types of forms. The following list is not exhaustive, but includes regularizations one may consider.
2. It may be appropriate to add a soft constraint on the parameter values in the objective function.
   - To account for prior knowledge (e.g., that the parameters have a bias).
   - To prefer simpler model classes that promote generalization.
   - To make an under-determined problem determined. (e.g., least squares with indeterminate $X^T X$.)
3. Dataset augmentation
4. Ensemble methods (i.e., essentially combining the output of several models).
5. Some training algorithms (e.g., stopping training early, dropout) can be seen as a type of regularization.

Stopping early

1. A straightforward (and popular) way to regularize is to constantly evaluate the training and validation loss on each training iteration, and return the model with the lowest validation error.
    - Requires caching the lowest validation error model.
    - Training will stop when after a pre-specified number of iterations, no model has decreased the validation error.
    - The number of training steps can be thought of as another hyper-parameter.
    - Validation set error can be evaluated in parallel to training.
    - It doesn't require changing the model or cost function.

1. A common (and simple to implement) type of regularization is to modify the cost function with a parameter norm penalty. This penalty is typically denoted as $\Omega(\theta)$ and results in a new cost function of the form:

$$\bar{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

with $\alpha \geq 0$

- α is a hyper-parameter that weights the contribution of the norm penalty.
- When $\alpha = 0$, there is no regularization.
- When $\alpha \to \infty$, the cost function is irrelevant and the model will set the parameters to minimize $\Omega(\theta)$.
- The choice of α can strongly affect generalization performance.
- When regularizing parameters, we typically do not regularize biases, since they do not introduce substantial variance to the estimator.

1. A common form of parameter norm regularization is to penalize the size of the weights, which is also called ridge regression or Tikhonov regularization. Let θ is union of w and bias.

$$\Omega(\theta) = \frac{1}{2} w^\top w$$

2. To prevent $\Omega(\theta)$ from getting large, L_2 regularization will cause the weights w to have small norm.

3. The new cost function is

$$\bar{J}(\theta) = J(\theta) + \frac{\alpha}{2} w^\top w$$

4. Its gradient equals to

$$\nabla_w \bar{J}(\theta) = \nabla_w J(\theta) + \alpha w$$

5. Parameters are updated using

$$w^{(t+1)} = (1 - \eta\alpha) w^{(t)} - \eta \nabla_w J(\theta)$$

1. In linear regression, the least squares solution $w = (X^\top X)^{-1} X^\top y$ becomes:

$$w = (X^\top X + \alpha I)^{-1} X^\top y$$

2. In linear regression, as M increases, the magnitude of the coefficients typically gets larger.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

3. The weights are updated using error-back-propagation algorithm when the input sample x is fed to the network.

Extensions of L_2 regularization

1. Other related forms of L_2 regularization include:

   1.1 Instead of a soft constraint that w be small, one may have prior knowledge that w is close to some value b. Then, the regularizer may take the form:

   $$\Omega(\theta) = \|w - b\|_2.$$

   1.2 One may have prior knowledge that two parameters, $w^{(1)}$ and $w^{(2)}$, must be close to each other. Then, the regularizer may take the form:

   $$\Omega(\theta) = \|w^{(1)} - w^{(2)}\|_2.$$

L_1 regularization

1. L_1 regularization defines the parameter norm penalty as

$$\Omega(\theta) = \|w\|_1$$

2. This penalty also causes the weights to be small.
3. The new cost function is

$$\bar{J}(\theta) = J(\theta) + \alpha\|w\|_1$$

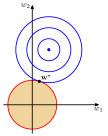
4. Its gradient equals to

$$\nabla_w \bar{J}(\theta) = \nabla_w J(\theta) + \alpha sign(w)$$

5. Empirically, this typically results in sparse solutions where $w_i = 0$ for several i.
6. This may be used for feature selection, where features corresponding to zero weights may be discarded.

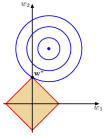
- L_2-Regularization has Closed form solution and can be solved in polynomial time

$$J(g) = \frac{1}{2}\sum_{i=1}^{m}(t_i - g(x_i))^2 + \frac{\lambda}{2}\|W\|^2.$$



- L_1-Regularization can be approximated in polynomial time

$$J(g) = \frac{1}{2}\sum_{i=1}^{m}(t_i - g(x_i))^2 + \lambda\|W\|.$$



- L_0-Regularization is NP-complete optimization problem

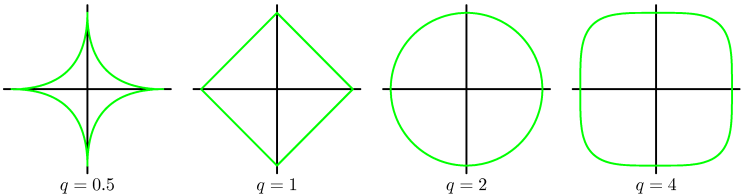
$$J(g) = \frac{1}{2}\sum_{i=1}^{m}(t_i - g(x_i))^2 + \lambda\sum_{j=1}^{M-1}\delta(w_j \neq 0).$$

The L_0-norm represents the optimal subset of features needed by a Regression model.

- A more general regularizer is sometimes used, for which the regularized error takes the form

$$J(g) = \frac{1}{2}\sum_{i=1}^{m}\left(t_i - g(x_i)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M-1}|w_j|^q.$$

- When $q = 2$, it is called Ridge regularizer
- When $q = 1$, it is called Lasso regularizer



Regularization and prior knowledge

- Assume that w has distribution of

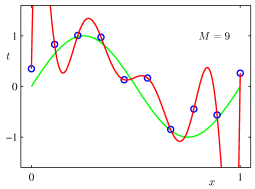
$$\mathbb{P}(w) = \mathcal{N}(0, \sigma^2 I_D).$$

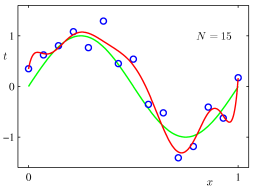
- Posterior density of w given set S results in L_2-regularization.
- Assume that w has isotropic Laplace distribution. Posterior density of w given set S results in L_1-regularization.
- What about other types of regularizes?

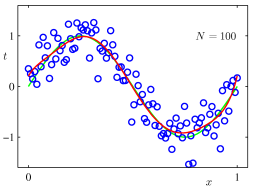
Dataset augmentation

Dataset augmentation

1. Can the more training data prevent the model from over-fitting.
2. For a given model complexity, the over-fitting problem become less severe as the size of the data set increases.







3. Best way to make a ML model to generalize better is to train it on more data
4. In practice amount of data is limited
5. Get around the problem by creating synthesized data
6. For some ML tasks it is straightforward to synthesize data

Dataset augmentation for classification

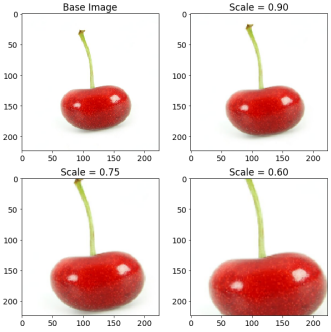
1. Data augmentation is easiest for classification
   - Classifier takes high-dimensional input x and summarizes it with a single category identity y
   - Main task of classifier is to be invariant to a wide variety of transformations
2. Generate new samples (x, y) just by transforming inputs
3. Approach not easily generalized to other problems

Example (Dataset augmentation for object recognition)

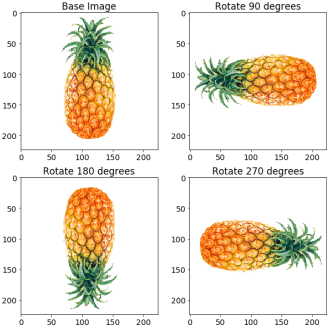
- Data set augmentation is very effective for the classification problem of object recognition
- Images are high-dimensional and include a variety of variations, may easily simulated
- Translating the images a few pixels can greatly improve performance
- Rotating and scaling are also effective
  - Considering flipping between b and d.
  - Considering rotation between 6 and 9.

Dataset augmentation for Image processing

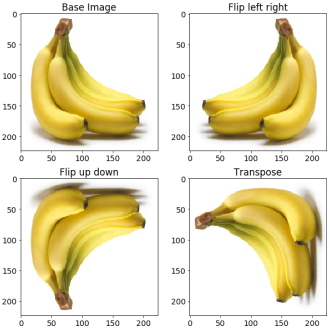
Scaling



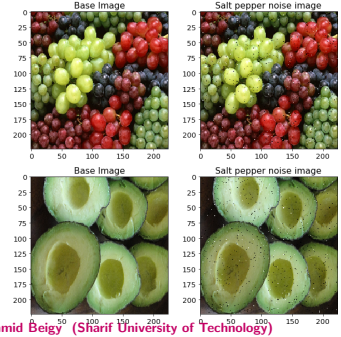
Rotation



Flipping

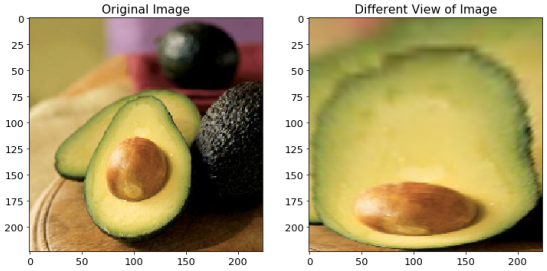


Adding Salt and Pepper noise

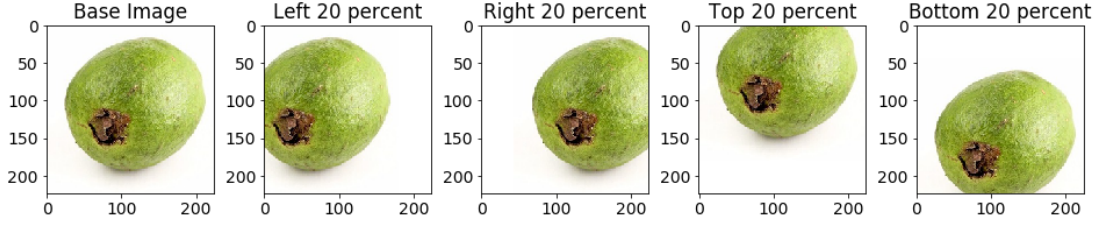


Lighting condition

Perspective transform



Translation



The result of dataset augmentation on Caltech101 dataset (Taylor and Nitschke 2018).

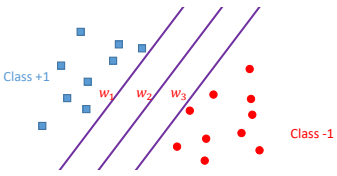
	Top-1 Accuracy	Top-5 Accuracy
Baseline	$48.13 \pm 0.42\%$	$64.50 \pm 0.65\%$
Flipping	$49.73 \pm 1.13\%$	$67.36 \pm 1.38\%$
Rotating	$50.80 \pm 0.63\%$	$69.41 \pm 0.48\%$
Cropping	$\mathbf{61.95 \pm 1.01\%}$	$\mathbf{79.10 \pm 0.80\%}$
Color Jittering	$\mathbf{49.57 \pm 0.53\%}$	$67.18 \pm 0.42\%$
Edge Enhancement	$49.29 \pm 1.16\%$	$66.49 \pm 0.84\%$
Fancy PCA	$49.41 \pm 0.84\%$	$\mathbf{67.54 \pm 1.01\%}$

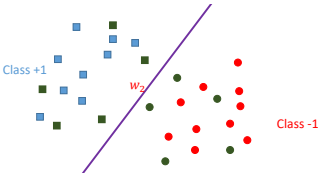
Top-1 score is the number of times the highest probability is associated with the correct target over all testing images.

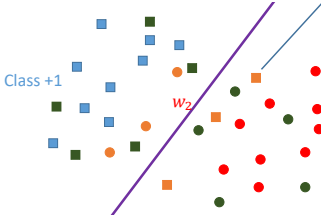
Top-5 score is the number of times the correct label is contained within the 5 highest probabilities.

Noise injection

1. Noise injection to the input can be seen as a form of data augmentation.
2. Neural networks are robust to noise.
3. The robustness of neural networks can be improved by adding noise to inputs and outputs of hidden units.
4. Caution for using noise







1. Easy data augmentation techniques (EDA) consists of four simple but powerful operations (Wei and Zou 2019).

   Synonym replacement (SR) Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.

   Random insertion (RI) Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this n times.

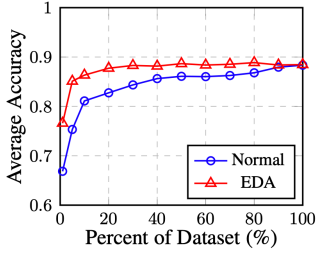
   Random swap (RS) Randomly choose two words in the sentence and swap their positions. Do this n times.

   Random deletion (RD) For each word in the sentence, randomly remove it with probability p.

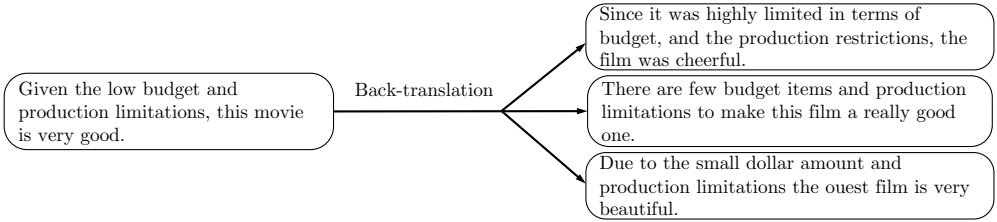
Sentences generated using EDA rules

Operation	Sentence
None	A sad, superior human comedy played out on the back roads of life.
SR	A ***lamentable***, superior human comedy played out on the ***backward*** road of life.
RI	A sad, superior human comedy played out on ***funniness*** the back roads of life.
RS	A sad, superior human comedy played out on ***roads*** back ***the*** of life.
RD	A sad, superior human out on the roads of life.

Average performance on 5 datasets w/wo EDA.



1. Unsupervised data augmentation techniques (UDA) translating an existing example x in language A into another language B and then translating it back into A to obtain an augmented example $\hat{x}$ (Xie et al. 2020).



Dataset augmentation

1. For more data augmentation techniques, please read the following papers.
   - Image data(Shorten and Khoshgoftaar 2019):
     https://link.springer.com/article/10.1186/s40537-019-0197-0.
   - Time series data(Wen et al. 2020).
   - EEG signals(Lashgari, Lianga, and Maoz 2020).
   - Text data(Ratner et al. 2017; Sahin and Steedman 2018; Wei and Zou 2019).
2. Data augmentation instead of explicit regularization(Hernández-García and König 2018).

Noise robustness

Adding noise is equivalent to weight decay

1. Suppose we add noise to input: $x + \epsilon$
2. Suppose the hypothesis is $h(x) = w^T x$, noise is $\epsilon \sim \mathcal{N}(0, \lambda I)$
3. Before adding noise, the loss is

$$J(\theta) = \mathbb{E}_{x,y}(f(x) - y)^2$$

4. After adding noise, the loss is

$$J(\theta) = \mathbb{E}_{x,y,\epsilon}(f(x + \epsilon) - y)^2$$

5. Simplifying the above equation yields to

$$J(\theta) = \mathbb{E}_{x,y,\epsilon}(f(x) - y)^2 + \lambda \|w\|^2$$

6. This is equivalent to weight decay (L_2 regularization)

Adding noise to labels

1. Many datasets has mistakes in label y.
2. In this case maximizing $-\log p(y|x)$ is harmful.
3. We can assume that for small constant ϵ, the training label is correct with probability of $(1-\epsilon)$ incorrect otherwise.
4. Extending the above case to k output values is label smoothing (Müller, Kornblith, and Hinton 2019).
5. For more information, please read (Song et al. 2020).

Adding noise to weights

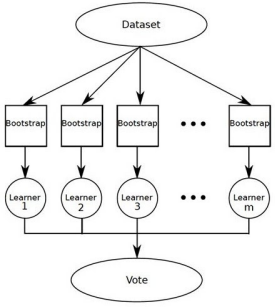
1. This technique primarily used with RNNs
2. This can be interpreted as a stochastic implementation of Bayesian inference over the weights
3. Adding noise to weights is a practical, stochastic way to reflect this uncertainty
4. Noise applied to weights is equivalent to traditional regularization, encouraging stability.

Bagging & Dropout

Bagging

1. It is short for Bootstrap Aggregating
2. It is a technique for reducing generalization error by combining several models

   Idea is to train several models separately, then have all the models vote on the output for test examples
3. This strategy is called model averaging
4. Techniques employing this strategy are known as ensemble methods.
5. Model averaging works because different models will not make the same mistake



Bagging

Consider the set of k regression models

1. Each model i makes error ϵ_i on each example
2. Errors drawn from a zero-mean multivariate normal with variance $\mathbb{E}[\epsilon_i^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$
3. Error of average prediction of all ensemble models: $\frac{1}{k} \sum_i \epsilon_i$
4. Expected squared error of ensemble prediction is

$$\mathbb{E}\left[\frac{1}{k} \sum_i \epsilon_i\right]^2 = \frac{1}{k} v + \frac{k-1}{k} c$$

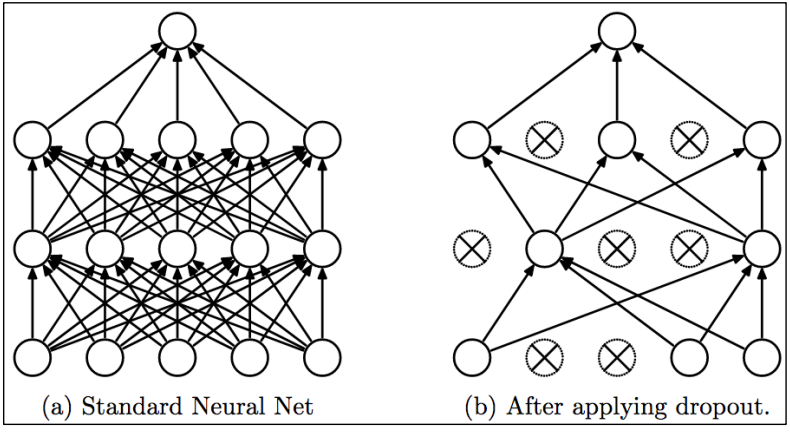
5. If errors are perfectly correlated, $c = v$, and mean squared error reduces to v, so model averaging does not help.
6. If errors are perfectly uncorrelated and $c = 0$, expected squared error of ensemble is only $\frac{v}{k}$ and Ensemble error decreases linearly with ensemble size

Dropout

1. Bagging is a method of averaging over several models to improve generalization
2. How do you apply the bagging to neural networks?
3. Impractical to train many neural networks since it is expensive in time and memory
4. Dropout makes it practical to apply bagging to very many large neural networks
5. It is a method of bagging applied to neural networks
6. Dropout is an inexpensive but powerful method of regularizing a broad family of models

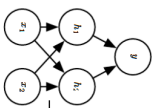
Dropout

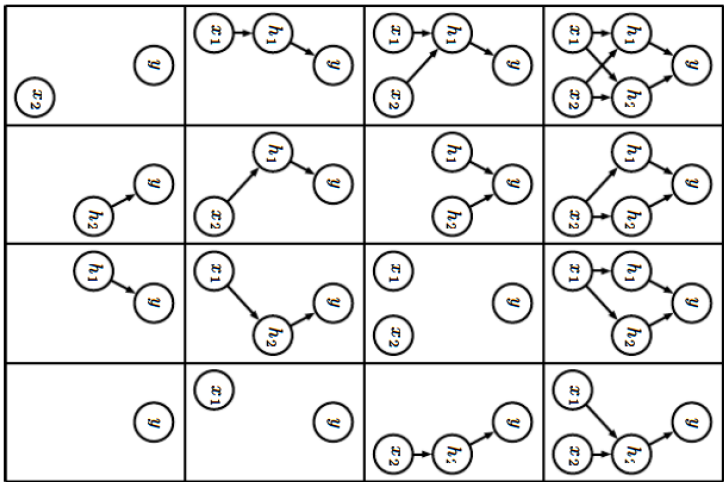
1. Dropout trains an ensemble of all subnetworks
2. Sub-networks formed by removing non-output units from an underlying base network
3. We can effectively remove units by multiplying its output value by zero



Dropout as bagging

1. In bagging we define k different models, construct k different data sets by sampling from the dataset with replacement, and train model i on dataset i
2. Dropout aims to approximate this process, but with an exponentially large number of neural networks





Mask for dropout training

1. To train with dropout we use mini-batch based learning algorithm that takes small steps such as SGD
2. At each step randomly sample a binary mask
3. Probability of including a unit is a hyper-parameter:

   0.5 for hidden units and 0.8 for input units
4. We run forward and backward propagation as usual

Dropout prediction

1. Each sub-model defines mask vector μ defines a probability distribution $p(y|x, \mu)$
2. Dropout prediction is
$$\sum_{\mu} p(y|x, \mu)$$
3. It is intractable to evaluate due to an exponential number of terms
4. We can approximate inference using sampling
5. By averaging together the output from many masks
6. 10-20 masks are sufficient for good performance
7. Even better approach, at the cost of a single forward propagation

Weight Initialization

Weight Initialization

1. Set all the initial weights to zero
   - Derivative with respect to loss function is the same for every w, for the same input pattern.
   - Weights have the same values in the subsequent iteration, for the same input pattern.
   - This makes the hidden units symmetric and continues for all the n iterations you run.
2. Set all the initial weights to small random numbers following standard normal distribution. This leads the following issues
   - Vanishing gradients: For any activation function, $|\frac{\partial J}{\partial w}|$ will get smaller and smaller as backpropagate error in layers.
   - Exploding gradients: For any activation function, $|\frac{\partial J}{\partial w}|$ will get larger and larger as backpropagate error in layers.
3. Sparse initialization. Set all the initial weights to zero and randomly connect each neuron to some of its inputs.
4. Set biases to zero.

Weight Initialization

1. Variance of neuron's output grows with the number of its inputs, n. One solution is to normalize the variance to 1, by scaling the weights by $\sqrt{n}$. Xavier weight initialization initializes weights as

$$w \sim U\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right)$$

2. Normalized Xavier weight initialization initializes weights as

$$w \sim U\left(-\frac{\sqrt{6}}{\sqrt{n+m}}, \frac{\sqrt{6}}{\sqrt{n+m}}\right)$$

   m is the number of outputs from the layer (the number of nodes in the current layer).

3. **In practice:** Use ReLU activation function and set weights to (He et al. 2015).

$$w \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

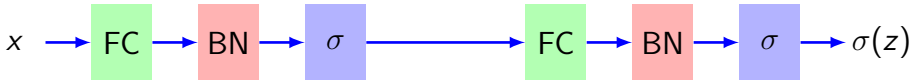
Normalization methods

Normalization methods

1. Normalization prepares data to change features in the dataset to a common scale when the features have different ranges.
2. There are several normalization approaches, which can be used in deep learning.
   - Data normalization: converting the data to zero mean and unit variance one.
   - Batch normalization
   - Layer normalization
   - Group normalization
   - Instance normalization
   - Weight normalization
   - Cosine normalization
3. Advantages of using normalization
   - Contribution of all features become almost the same (features with large/small ranges).
   - Internal covariate shift will be reduced. Internal covariate shift is the change in the distribution of network activations due to the change in network parameters during training.
   - The loss surface becomes smoother.
   - Optimization becomes faster, because weights don't explode all over the place and is restricted to a certain range.

Batch normalization

1. Batch normalization normalizes activations in a network (Ioffe and Szegedy 2015).



2. Batch normalization computes the mean and variance of that feature in every mini-batch.

$$\mu_B = \frac{1}{K}\sum_{k=1}^{K} x_k \qquad \sigma_B^2 = \frac{1}{K}\sum_{k=1}^{K}(x_k - \mu_B)^2$$

3. Then, it normalizes every feature using this mean and variance (ϵ is for numerical stability).

$$\hat{x}_k = \frac{x_k - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

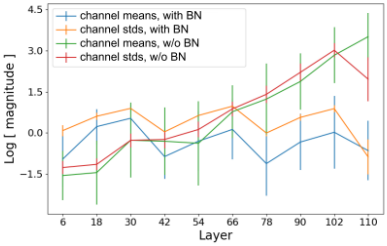
4. But increasing the magnitude of weights may improve network performance.

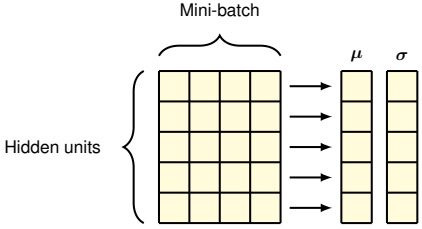
$$\hat{y}_k = \gamma \hat{x}_k + \beta = BN_{\gamma,\beta}(x_k)$$

γ and β are learnable parameters.

Batch normalization

1. BN reduces internal covariate shift and accelerates the training of a deep neural network.
2. BN reduces the dependence of gradients on the scale of parameters or of their initial values, which results in higher learning rates without the risk of divergence.
3. BN makes it possible to use saturating nonlinearities by preventing the network from getting stuck in the saturated modes.
4. BN reduces the common problem of vanishing gradients.





5. In training phase, we normalize the samples to zero means and unit variance.
6. What happens in the testing phase?

1. In layer normalization, mean and variance are calculated for each individual sample across all channels and both spatial dimensions and is very useful for RNN (L. J. Ba, Kiros, and Hinton 2016).

$$\mu_n = \frac{1}{CHW}\sum_{i=1}^{C}\sum_{j=1}^{H}\sum_{k=1}^{W} a^{ijk}$$

$$\sigma_n^2 = \sqrt{\frac{1}{CHW}\sum_{i=1}^{C}\sum_{j=1}^{H}\sum_{k=1}^{W}(a^{ijk} - \mu_n)^2}$$

2. Then, it normalizes every feature using this mean and variance (ϵ is for numerical stability.).

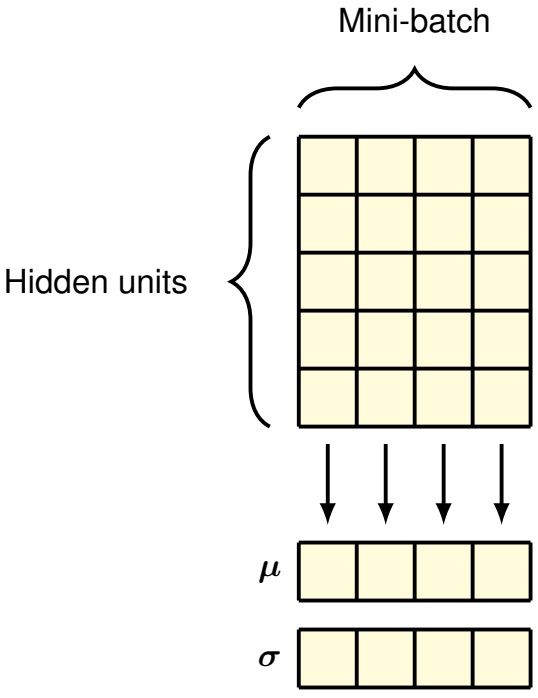
$$\hat{a}^k = \frac{a_k - \mu_k}{\sqrt{\sigma_n^k + \epsilon}}$$

3. But increasing the magnitude of weights may improve network performance.

$$\hat{y}^k = \gamma\hat{a}^k + \beta = LN_{\gamma,\beta}(a^k)$$

γ and β are learnable parameters.

Layer normalization (LN)



Group normalization (GN)

1. GN divides channels into groups and then computes group statistics over the group (Wu and He 2020).
2. GN computes the mean and variance of that feature in every group G.

$$\mu^g = \frac{1}{|G|} \sum_{k \in G} a^k$$

$$\sigma^g = \sqrt{\frac{1}{|G|} \sum_{k \in G} \left(a^k - \mu^g\right)^2}$$

3. Then, it normalizes every feature using this mean and variance (ϵ is for numerical stability.).

$$\hat{a}^k = \frac{a^k - \mu^g}{\sqrt{\sigma^{2,g} + \epsilon}}$$

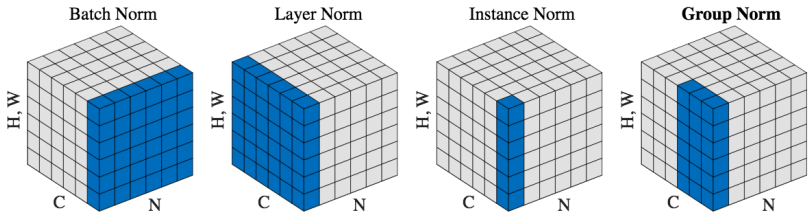
4. But increasing the magnitude of weights may improve network performance.

$$\hat{y}^k = \gamma \hat{a}^k + \beta = GN_{\gamma,\beta}(a^k)$$

γ and β are learnable parameters.

Instance normalization

1. IN normalizes each channel of each batch's image independently (Ulyanov, Vedaldi, and Lempitsky 2016).
2. For example, IN normalization is done across the width and height of a single feature map of a single example.
3. The goal is to normalize the contrast of the content image.



Weight normalization (WN) & Cosine normalization (CN)

1. In WN, the weight vectors is expressed as (Salimans and Kingma 2016).

$$W = \frac{g}{\|V\|} V$$

   g and V are learnable scalar and a learnable matrix, respectively.

2. CN normalizes both the weights and the input by replacing the classic dot product by a cosine similarity:

$$y = \sigma\left(\frac{\langle X, W\rangle}{\|W\| \times \|X\|}\right)$$

   σ is a nonlinear function.

3. For more information, please read (Huang et al. 2020).

Other approaches for increasing generalization

1. Multi-task learning
2. Semi-supervised learning
3. other ensemble methods such as boosting?

Reading

Readings

1. Chapter 7 of Deep Learning Book[2].
2. Chapters 8 and 9 of Deep Learning: Foundations and Concepts[3].

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. The MIT Press.
[3] Christopher M. Bishop and Hugh Bishop (2024). *Deep Learning: Foundations and Concepts*. Springer.

References i

- Ba, Lei Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". In: *CoRR* abs/1607.06450.
- Bishop, Christopher M. and Hugh Bishop (2024). *Deep Learning: Foundations and Concepts*. Springer.
- Dozat, Timothy (2016). *Incorporating Nesterov Momentum into Adam*. eprint: `http://cs229.stanford.edu/proj2015/054_report.pdf`.
- Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12.61, pp. 2121–2159.
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. The MIT Press.
- He, Kaiming et al. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. eprint: `arXiv:1502.01852`.
- Hernández-García, Alex and Peter König (2018). *Data augmentation instead of explicit regularization*. eprint: `arXiv:1806.03852`.
- Huang, Lei et al. (2020). "Normalization Techniques in Training DNNs: Methodology, Analysis and Application". In: *CoRR* abs/2009.12836.
- Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37, pp. 448–456.

References ii

- Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.
- Lashgari, Elnaz, Dehua Lianga, and Uri Maoz (2020). "Data augmentation for deep-learning-based electroencephalography". In: *Journal of Neuroscience Methods* 346.1, p. 108885.
- Müller, Rafael, Simon Kornblith, and Geoffrey E. Hinton (2019). "When does label smoothing help?" In: *Advances in Neural Information Processing Systems*, pp. 4696–4705.
- Nesterov, Y. (1983). "A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$". In: *Dokl. akad. nauk Sssr (translated as Soviet.Math.Docl.)* 269, pp. 543–547.
- Qian, N. (1999). "On the momentum term in gradient descent learning algorithms". In: *Neural Networks* 12.1, pp. 145–151.
- Ratner, Alexander J. et al. (2017). "Learning to Compose Domain-Specific Transformations for Data Augmentation". In: *Advances in Neural Information Processing Systems*, pp. 3236–3246.
- Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2018). "On the Convergence of Adam and Beyond". In: *International Conference on Learning Representations*.
- Ruder, Sebastian (2016). *An overview of gradient descent optimization algorithms*. eprint: `arXiv:1609.04747`.

References iii

- Sahin, Gözde Gül and Mark Steedman (2018). "Data Augmentation via Dependency Tree Morphing for Low-Resource Languages". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Ed. by Ellen Riloff et al., pp. 5004–5009.
- Salimans, Tim and Diederik P. Kingma (2016). "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks". In: *Advances in Neural Information Processing Systems*.
- Shorten, Connor and Taghi M. Khoshgoftaar (2019). "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1, p. 60.
- Song, Hwanjun et al. (2020). "Learning from Noisy Labels with Deep Neural Networks: A Survey". In: *CoRR* abs/2007.08199. URL: https://arxiv.org/abs/2007.08199.
- Sun, Shiliang et al. (2019). *A Survey of Optimization Methods from a Machine Learning Perspective*. eprint: arXiv:1906.06821.
- Taylor, L. and G. Nitschke (2018). "Improving Deep Learning with Generic Data Augmentation". In: *Proceedings of IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1542–1547.
- Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky (2016). "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022.

- Wei, Jason W. and Kai Zou (2019). "EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks". In: *Proc. of the Conf. on Empirical Methods in Natural Language Processing and the 9th Int. Joint Conf. on Natural Language Processing, EMNLP-IJCNLP*, pp. 6381–6387.
- Wen, Qingsong et al. (2020). *Time Series Data Augmentation for Deep Learning: A Survey*. eprint: arXiv:2002.12478.
- Wu, Yuxin and Kaiming He (2020). "Group Normalization". In: *International Journal of Computer Vision* 128.3, pp. 742–755.
- Xie, Qizhe et al. (2020). "Unsupervised Data Augmentation for Consistency Training". In: *Advances in Neural Information Processing Systems (NeurIPS)*.
- Zeiler, M. D. (2012). *ADADELTA: An Adaptive Learning Rate Method*. eprint: arXiv:1212.5701.

Questions?