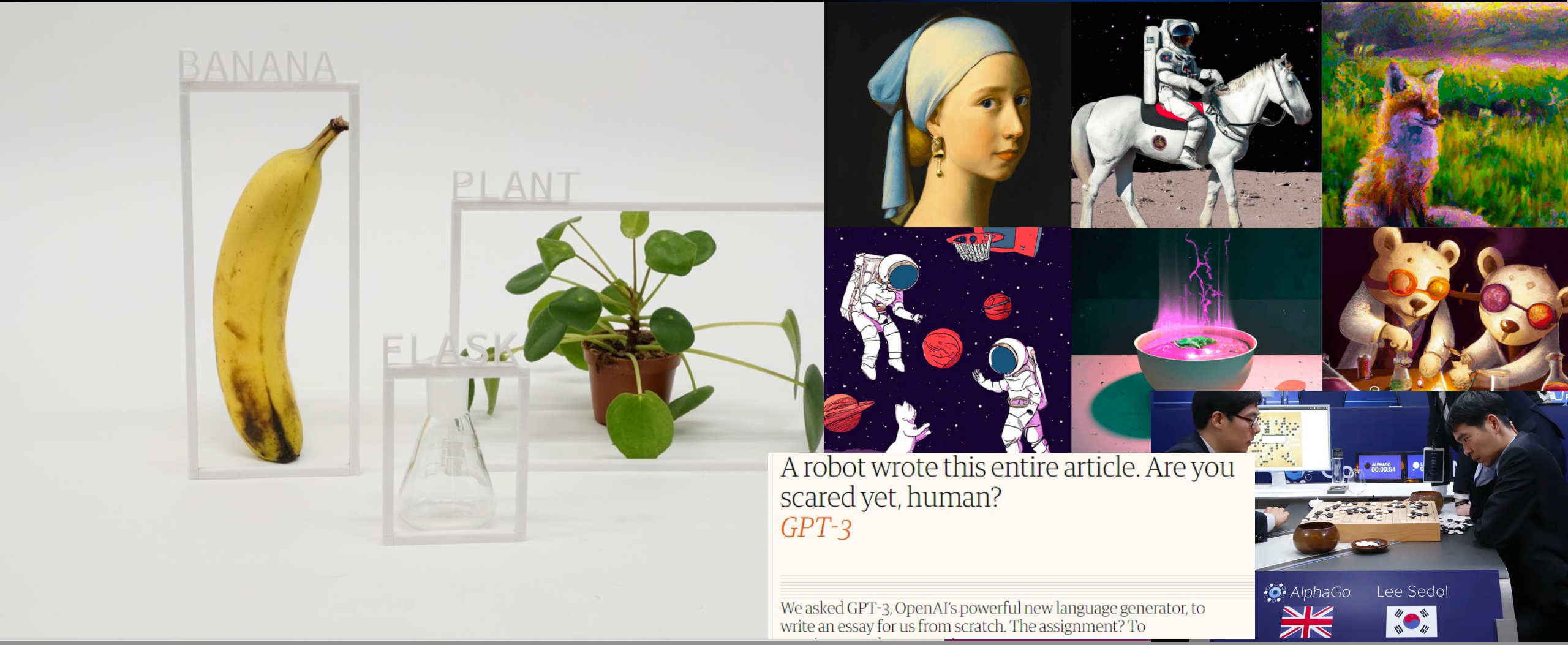
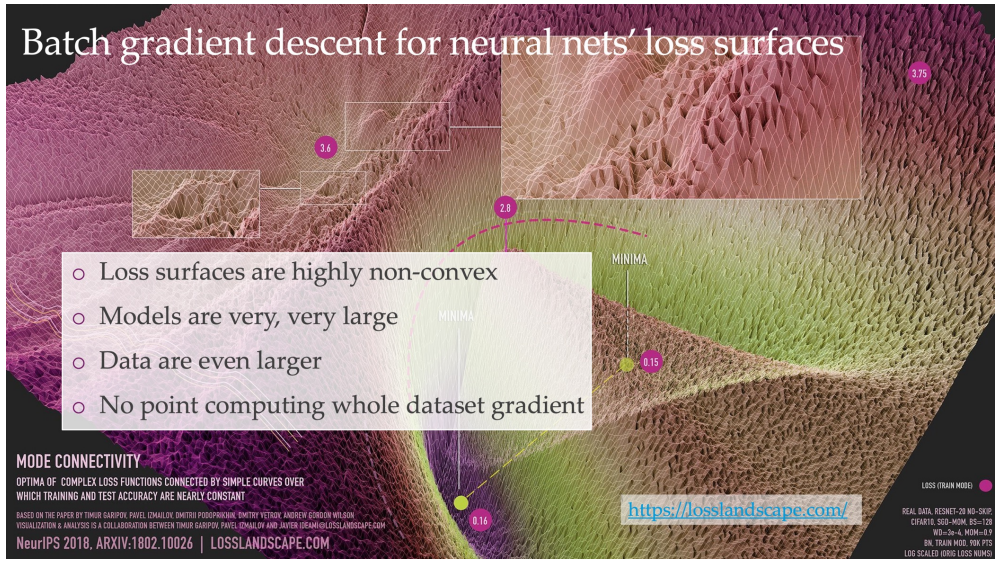


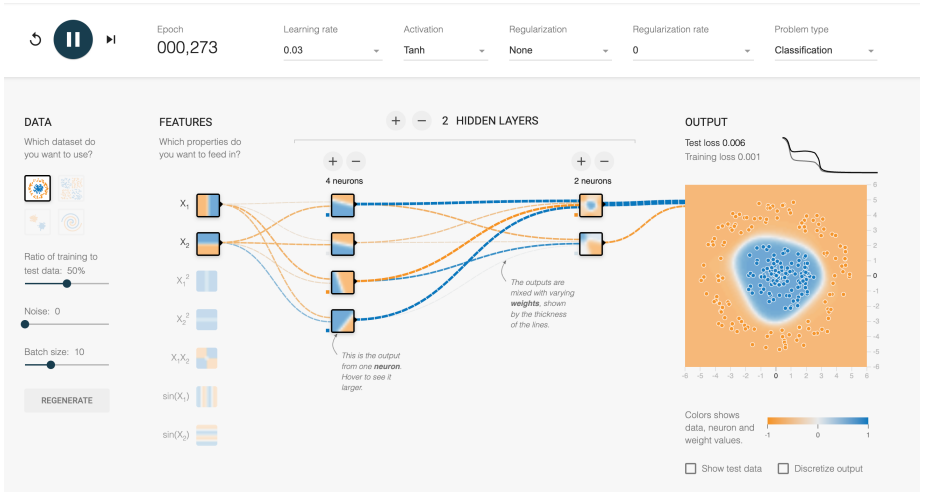
Lecture 4: Deep Learning Optimizations II

Deep Learning 1 @ UvA

Yuki M. Asano

Last time:



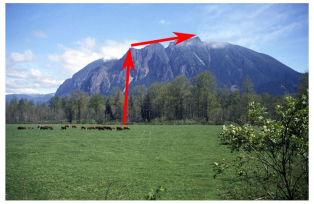


1. Ill-conditioning

- Consider the Hessian matrix H has an eigenvalue decomposition, its condition number is

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$$

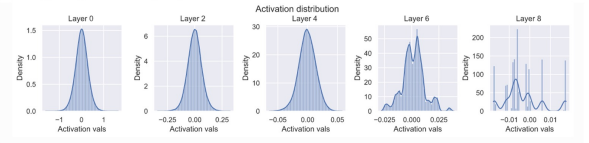
- This is the ratio of the magnitude of the largest (i) and smallest eigenvalue (j).
- The condition number measure show much the second derivatives differ from each other.
- With a poor (large) condition number, **gradient descent performs poorly.**
- It also makes it difficult to choose a good step size.



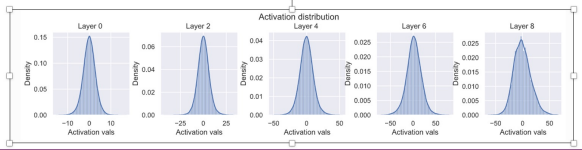
Gradient mostly points into 1 direction.
For n-dimension mountain --> n steps

Bad initialization can cause problems

Initializing weights in every layer with same constant variance → can diminish variance in activations



Initializing weights in every layer with increasing variance → can explode the variance in activations



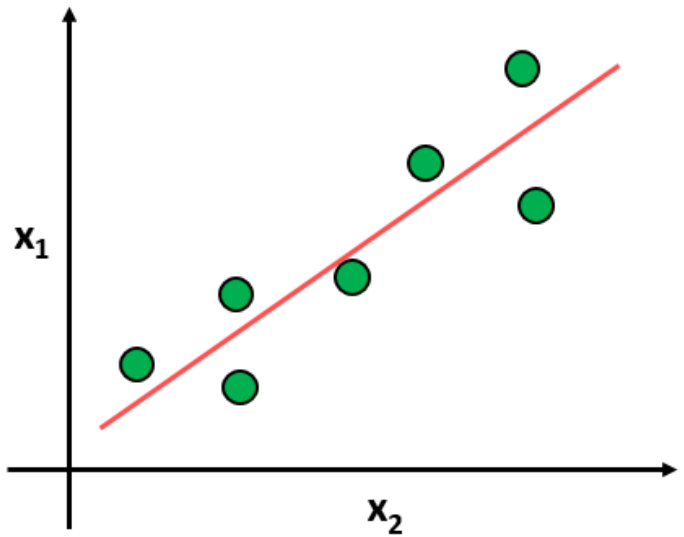
Lecture overview

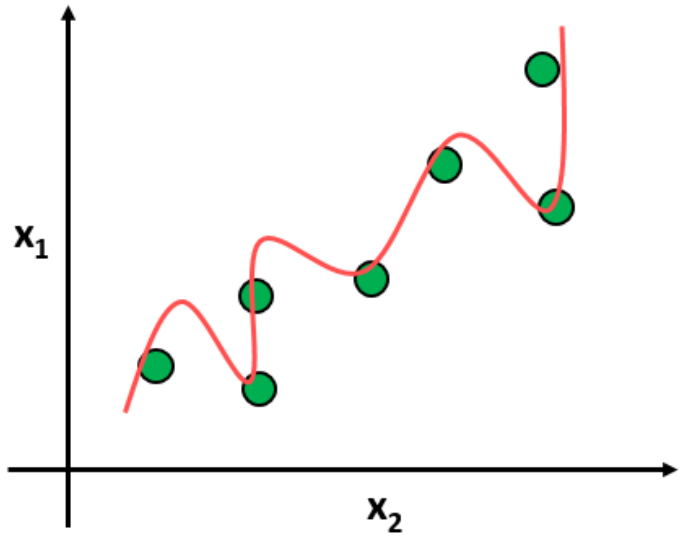
- Backpropagation
- In
- More
- Detail
- (kidding)

Lecture overview

- Normalization
- Regularization
- Hyperparameters

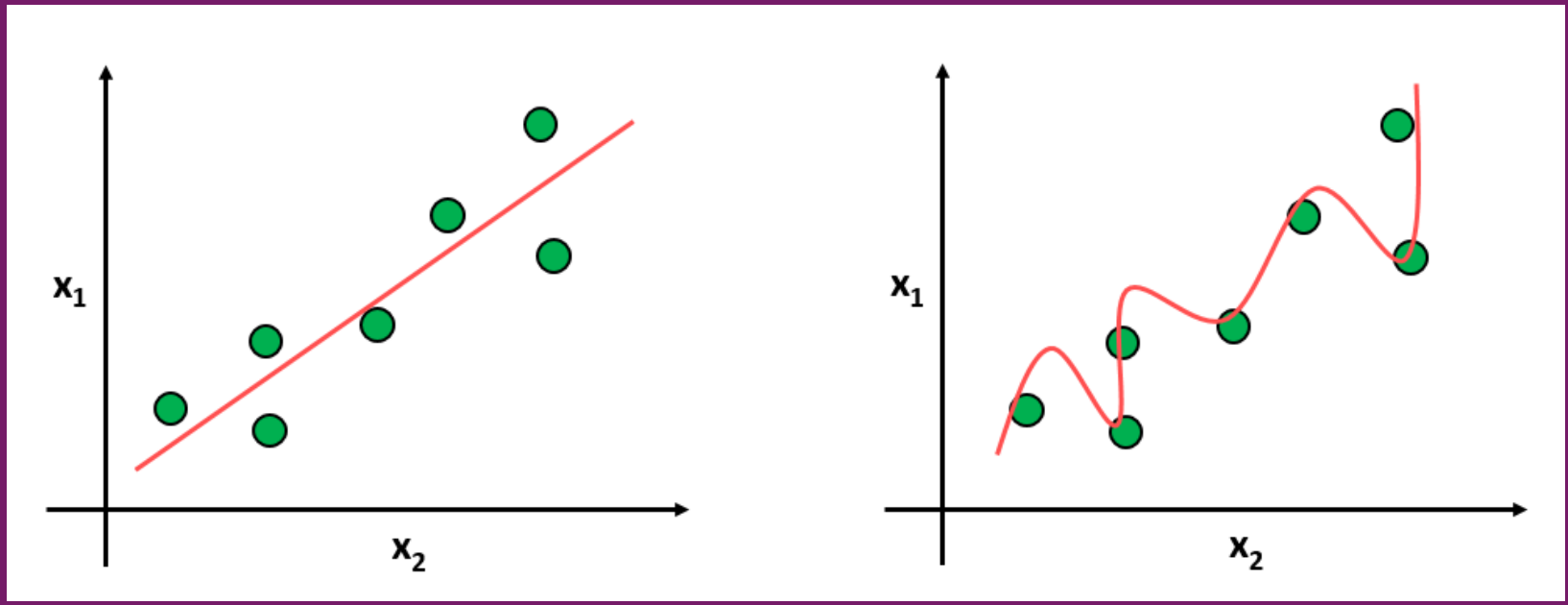
Regularization





Why might the left be better?

Why might the right be better?



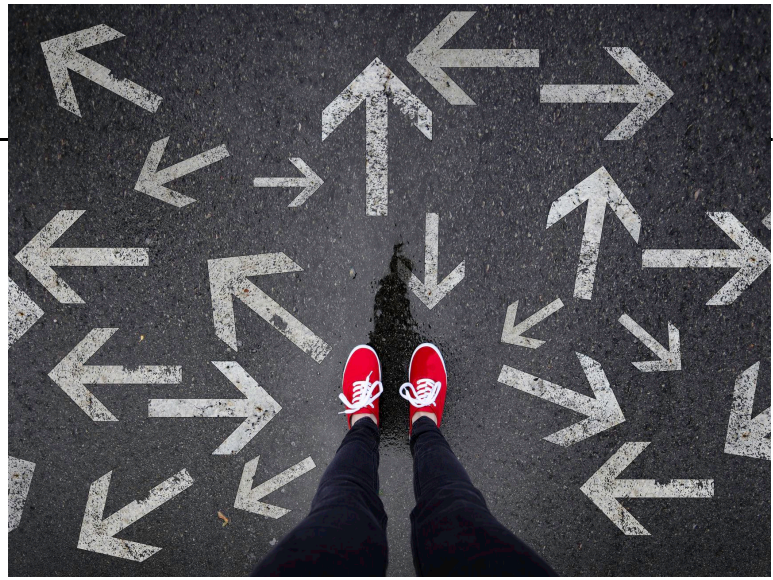
Digression: Gravitation

Newton:

$$F = G\frac{m_1 m_2}{r^2},$$

Einstein:

$$V(r) = \overbrace{-\frac{GMm}{r} + \frac{L^2}{2\mu r^2}}^{\textit{Newtonian Part}} \overbrace{-\frac{G(M+m)L^2}{c^2\mu r^3}}^{\textit{General Relativity Correction Term}}$$



What if nature is fundamentally non-linear / complicated?

Bias-variance tradeoff

- Point Estimation
  - The single "best" prediction of some quantity of interest, which can be a single parameter or a vector of parameters.
  - Point estimate of the relationship between input and target variables is referred to as function estimators.
  - A point estimator or statistic is any function of the data:

$$\hat{\boldsymbol{\theta}}_m = g(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$$

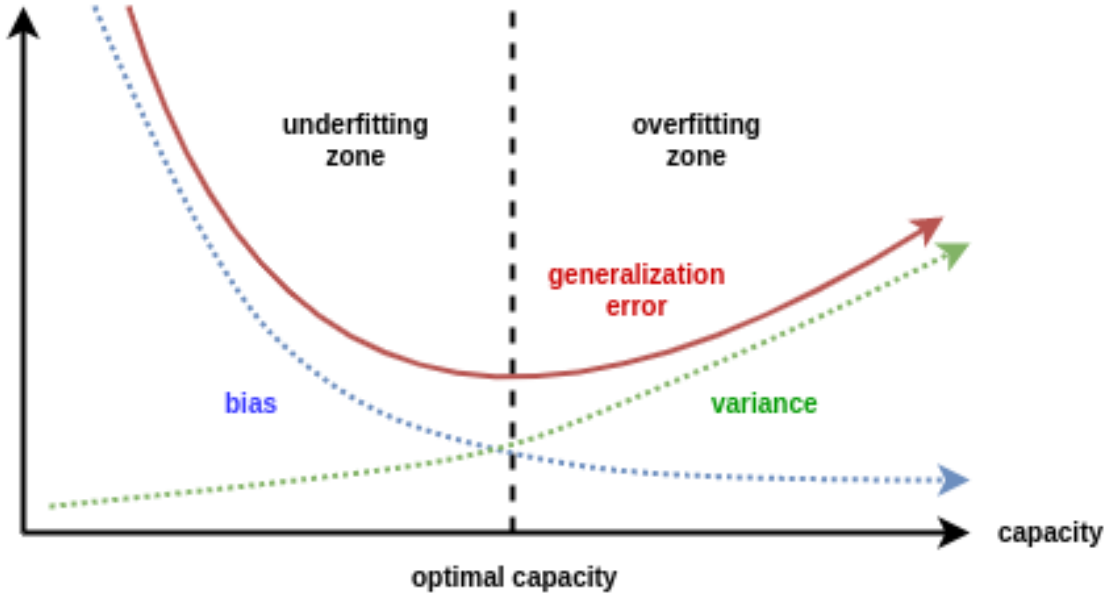
  where $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ are independent and identically distributed.

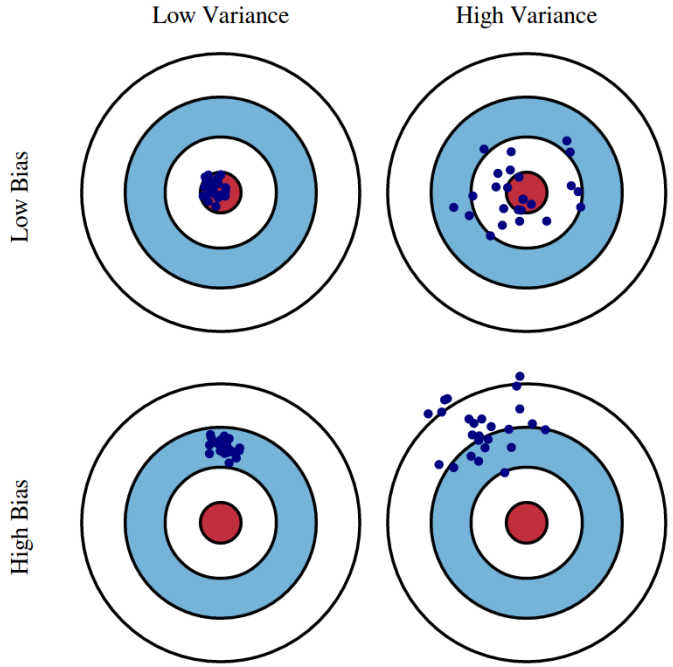
- A good estimator is a function whose output is close to the true underlying θ that generated the data

Bias-variance tradeoff

- Bias
  - “The difference between this estimator’s expected value and the true value of the parameter being estimated” .
  - The bias error is an error from erroneous assumptions in the learning algorithm.
- Variance
  - The amount that the estimate of the target function will change if different training data was used.
  - The variance is an error from sensitivity to small fluctuations in the training set.

Bias-variance tradeoff



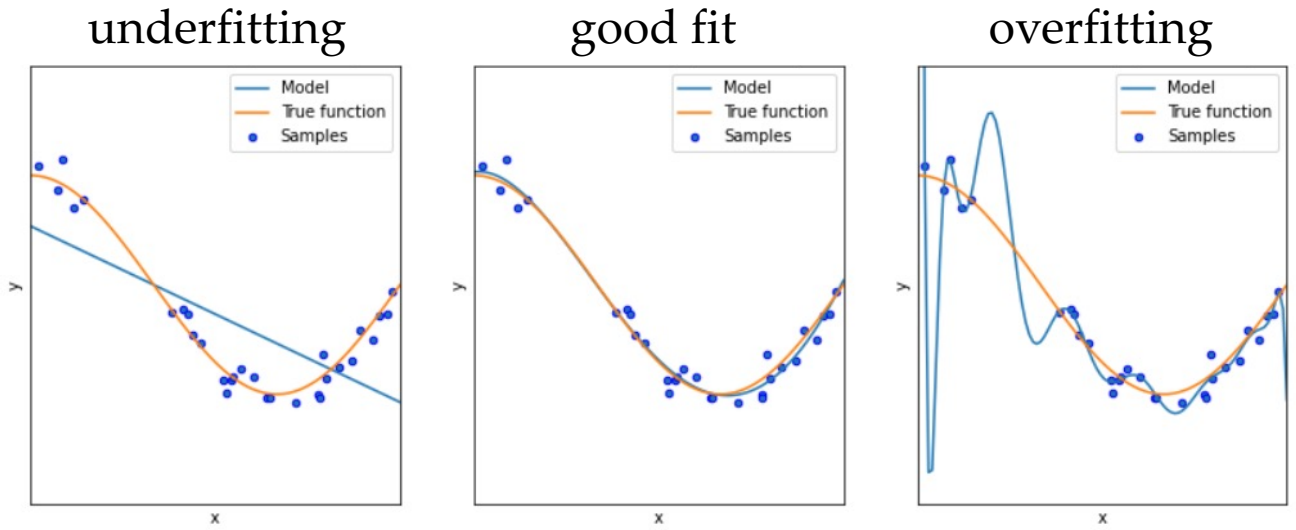


Link

- High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting), e.g., linear models.
- High variance may result from an algorithm modelling the random noise in the training data (overfitting), e.g., nonlinear models.

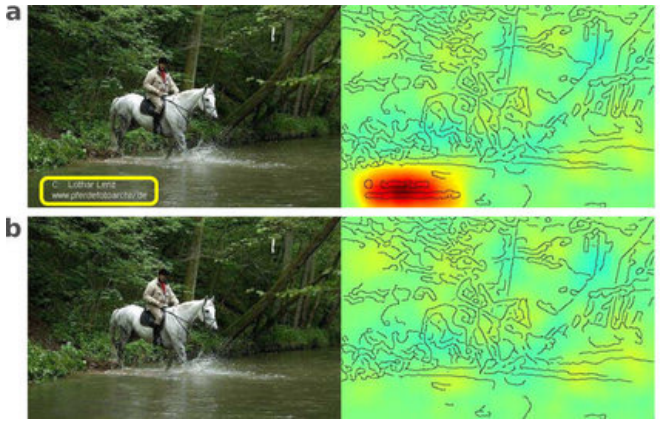
Overfitting

- One of the most important aspects when training neural networks is avoiding overfitting.
  - Overfitted models fail on new data from the same problem domain.
  - Caused by noise in the training data that the neural network picks up during training and learns it as an underlying concept of the data.



Overfitting

- *Why does the neural network pick up that noise in the first place?*
  - the complexity of this network is too high (for the problem).
  - is able to pick up and learn patterns (noise) in the data.
  - try to model each data sample, while not learning underlying true function
  - New arbitrary samples generated with the true function would have a high distance to the fit of the model (remember how non-linear neural networks are!)



Overfitting

- Neural networks typically have thousands, if not millions of parameters
  - Usually, the dataset size smaller than the number of parameters
- Overfitting is a grave danger
- Overfitting causes the neural network model to perform very well during training, but the performance gets much worse during inference time when faced with brand new data.
- To prevent overfitting or a high variance we must use something that is called *regularization*.

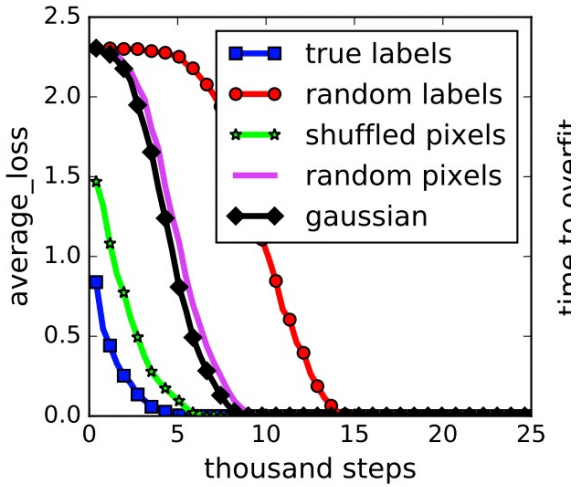
Overfitting: how "powerful" are neural networks?

Randomization tests. At the heart of our methodology is a variant of the well-known randomization test from non-parametric statistics (Edgington & Onghena, 2007). In a first set of experiments, we train several standard architectures on a copy of the data where the true labels were replaced by random labels. Our central finding can be summarized as:

Deep neural networks easily fit random labels.

More precisely, when trained on a completely random labeling of the true data, neural networks achieve 0 training error. The test error, of course, is no better than random chance as there is no correlation between the training labels and the test labels. In other words, by randomizing labels alone we can force the generalization error of a model to jump up considerably without changing the model, its size, hyperparameters, or the optimizer. We establish this fact for several different standard architectures trained on the CIFAR10 and ImageNet classification benchmarks. While simple to state, this observation has profound implications from a statistical learning perspective:

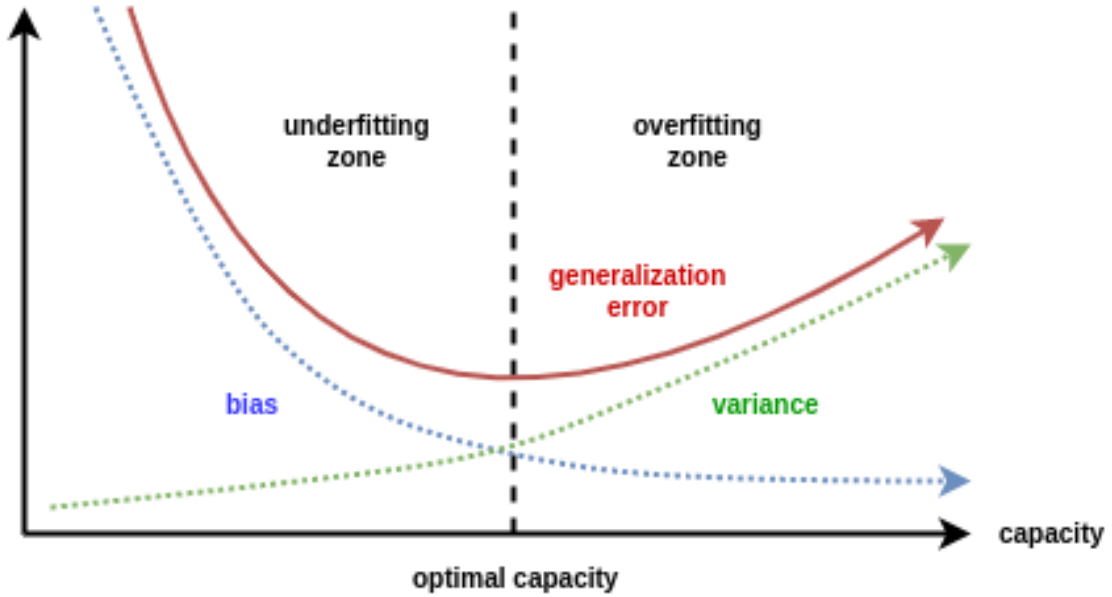
1. The effective capacity of neural networks is sufficient for memorizing the entire data set.
2. Even optimization on random labels remains easy. In fact, training time increases only by a small constant factor compared with training on the true labels.
3. Randomizing labels is solely a data transformation, leaving all other properties of the learning problem unchanged.



(a) learning curves

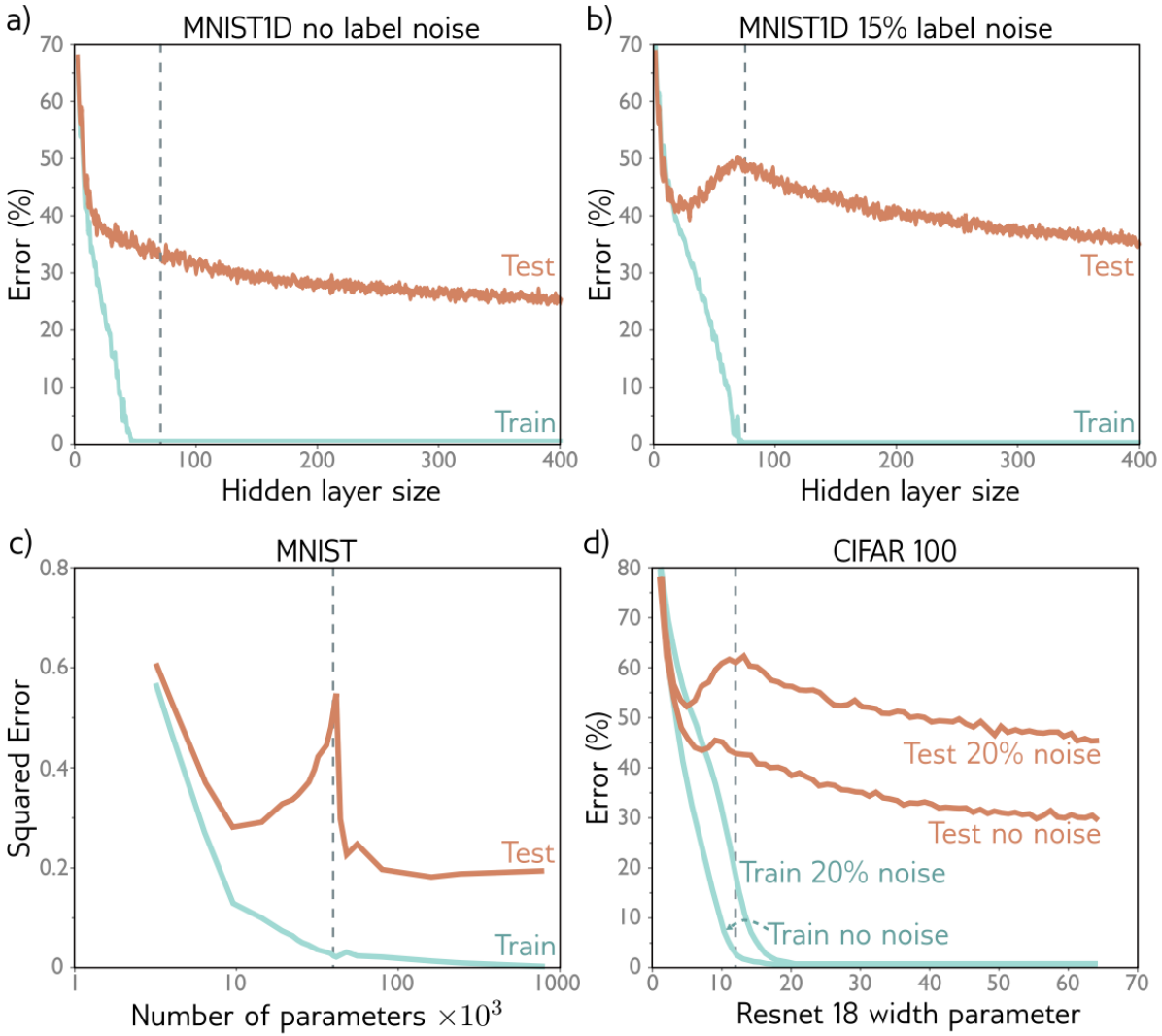
Understanding deep learning requires rethinking generalization. Zhang et al. 2016

This isn't the full story..



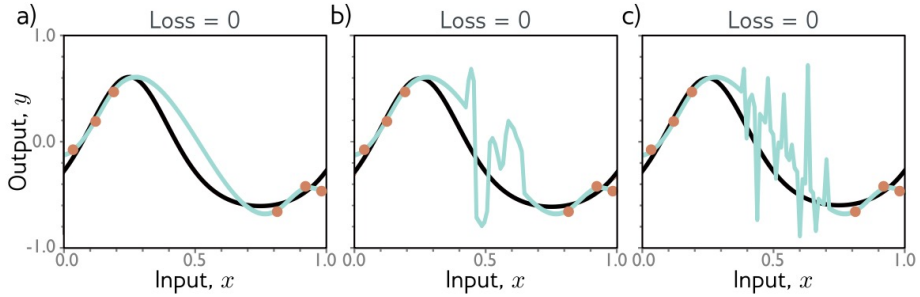
Link

Update 2019: Double Descent



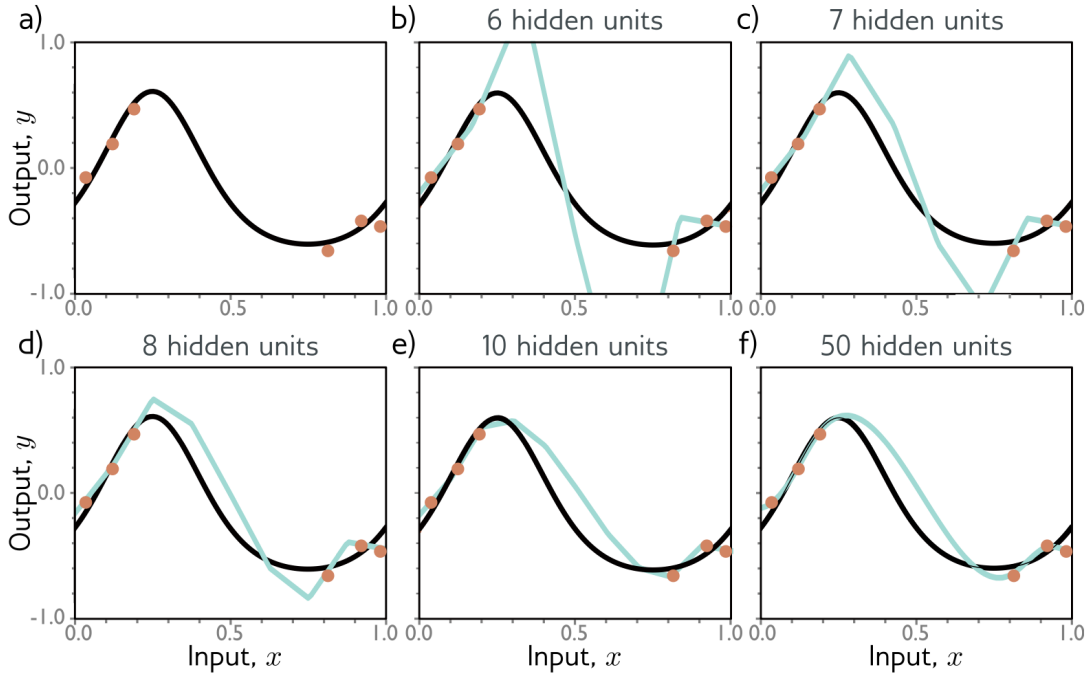
The MNIST-1D dataset has 40 dimensions and here 10K samples

- a) no "usual" bias-variance trade-off curve: easy
- b) "usual" bias-variance trade-off curve
  - … except test error goes down again and ends up lower?!
- Same for c) d).
- What's going on?
- Smoothness, regularisation

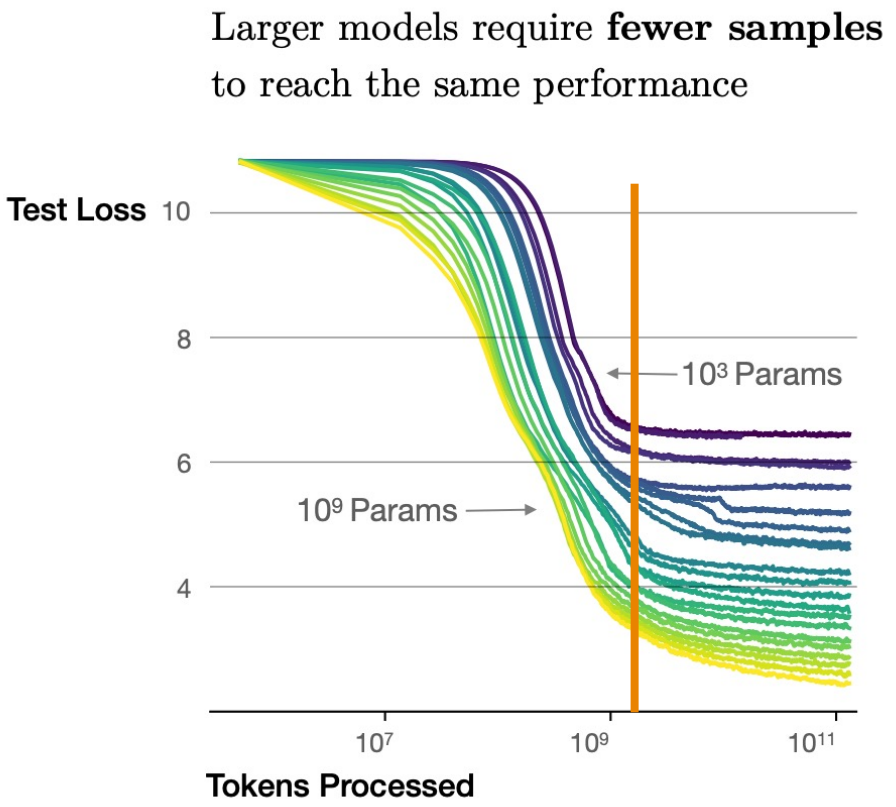


Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019) Reconciling modern machine-learning practice and the classical bias–variance trade- off. Proceedings of the National Academy of Sciences

Double-descent: Smoothness from bigger models



This matters, because we're seeing that larger and larger models are performing much better even at less compute ->



Scaling Laws for Neural Language Models. Kaplan et al. 2020

Double-descent in practice?

- In practice, we (so far) rarely see this
- Why?
  - Not in the double-descent regime
  - "Hoping" for more neurons to solve the problem doesn't solve it
  - Problems actually even more higher-dimensional because of augmentations
- So?
  - We need regularisation techniques (also used for those huge language models!)

Regularization

- Regularization refers to a set of different techniques that lower the complexity of a neural network model during training, and thus prevent the overfitting.
- Possible regularization methods
  - ℓ_2-regularization
  - ℓ_1-regularization
  - Early stopping
  - Dropout
  - *anything…*

1) ℓ_2-regularization

- The L_2 regularization is the most common type of all regularization techniques
  - commonly known as *weight decay* or *ridge regression* (in the linear case).
- The regularization term Ω is defined as the Euclidean Norm (or L2 norm) of the weight matrices
  - which is the sum over all squared weight values of a weight matrix.

$$\frac{1}{2}\sum_{l}\|w_l\|_2^2$$

- L2 regularization encourages the weight values towards zero
- A Gaussian prior on weights

1) ℓ_2-regularization

- The loss function with ℓ_2-regularization:

$$\mathrm{w}^* \leftarrow \arg\min_w \sum_{(x,y)\subseteq(X,Y)} \mathcal{L}(y, a_L(x; w_{1,\ldots,\mathrm{L}})) + \frac{\lambda}{2}\sum_l \|w_l\|_2^2$$

- The ℓ_2-regularization is added to the gradient descent update rule

$$w_{t+1} = w_t - \eta_t(\nabla_\theta \mathcal{L} + \lambda w_l) \Longrightarrow$$
$$w_{t+1} = (1 - \lambda\eta_t)w^{(t)} - \eta_t \nabla_\theta \mathcal{L}$$

- λ is usually about $10^{-1}, 10^{-2}$

"Weight decay", because weights get smaller

2) ℓ_1-regularization

- L1 regularization, also knows as Lasso regression
  - The sum of the absolute values of the weight parameters in a weight matrix

$$\frac{1}{2}\sum_{l}|w_l|$$

  - The regularization term does not scale linearly, contrary to L2 regularization, but it's a constant factor with an alternating sign.

  - A prior of an isotropic Laplacian distribution on weights.

2) ℓ_1-regularization

- ℓ_1-regularization is one of the most important regularization techniques

$$\mathrm{w}^* \leftarrow \arg\min_w \sum_{(x,y)\subseteq(X,Y)} \mathcal{L}(y, a_L(x; w_{1,\ldots,\mathrm{L}})) + \frac{\lambda}{2}\sum_l |w_l|$$

- Also ℓ_1-regularization is added to the gradient descent update rule

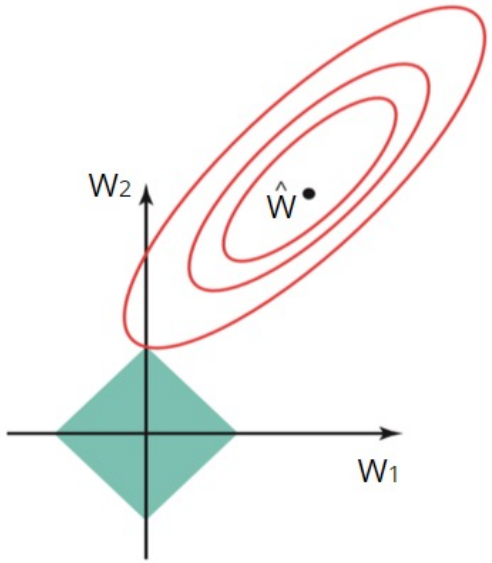
$$w_{t+1} = w_t - \eta_t \left(\nabla_\theta \mathcal{L} + \lambda \frac{w^{(t)}}{\mathrm{sgn}(w^{(t)})} \right)$$

- ℓ_1-regularization → sparse weights
  - $\lambda \nearrow$ → more weights become 0

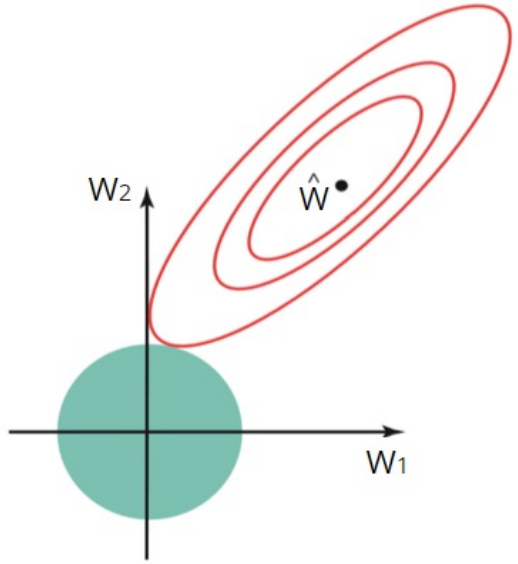
Sign function

Why do L1 and L2 Regularizations work?

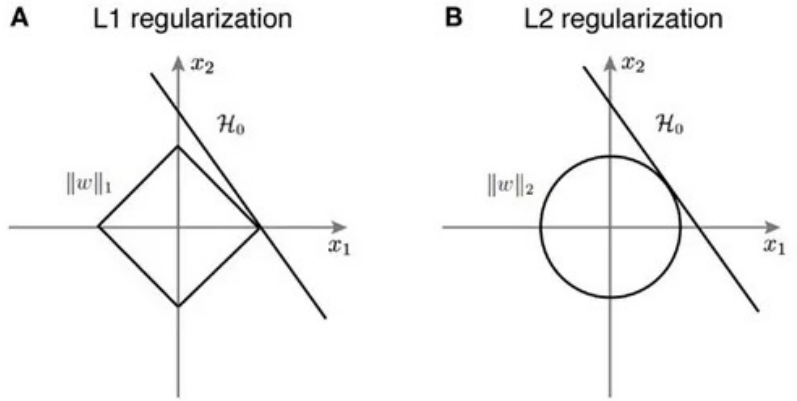
- The estimates of W_1 and W_2 are given by the first point where the ellipse intersects with the green constraint area.
- The other green constrained parts have worse losses



ℓ_1-regularization



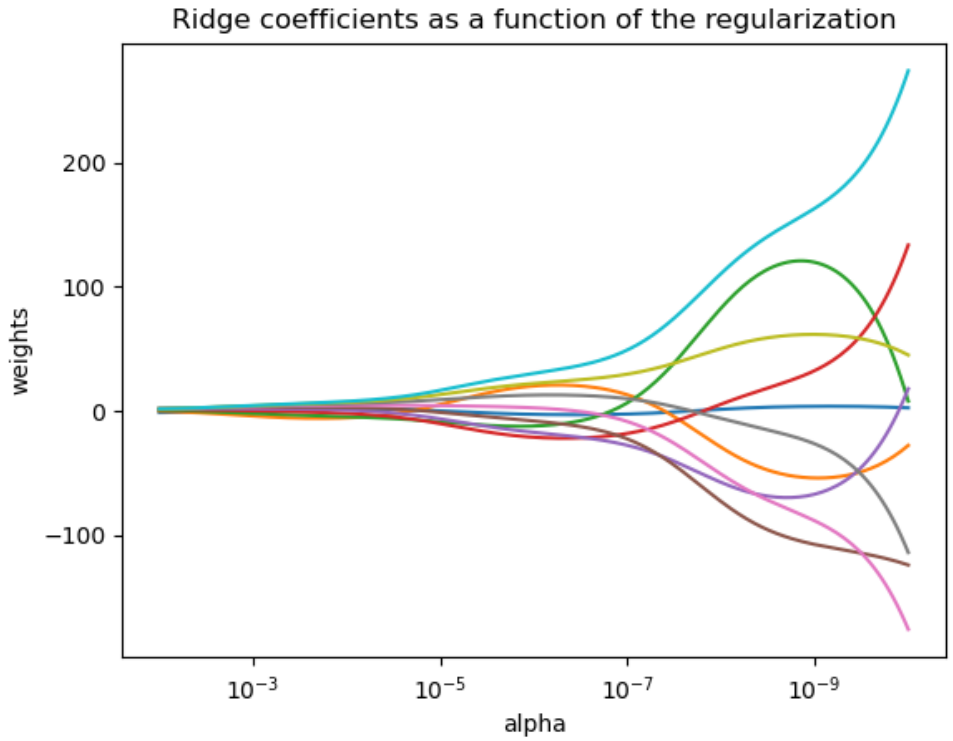
ℓ_2-regularization



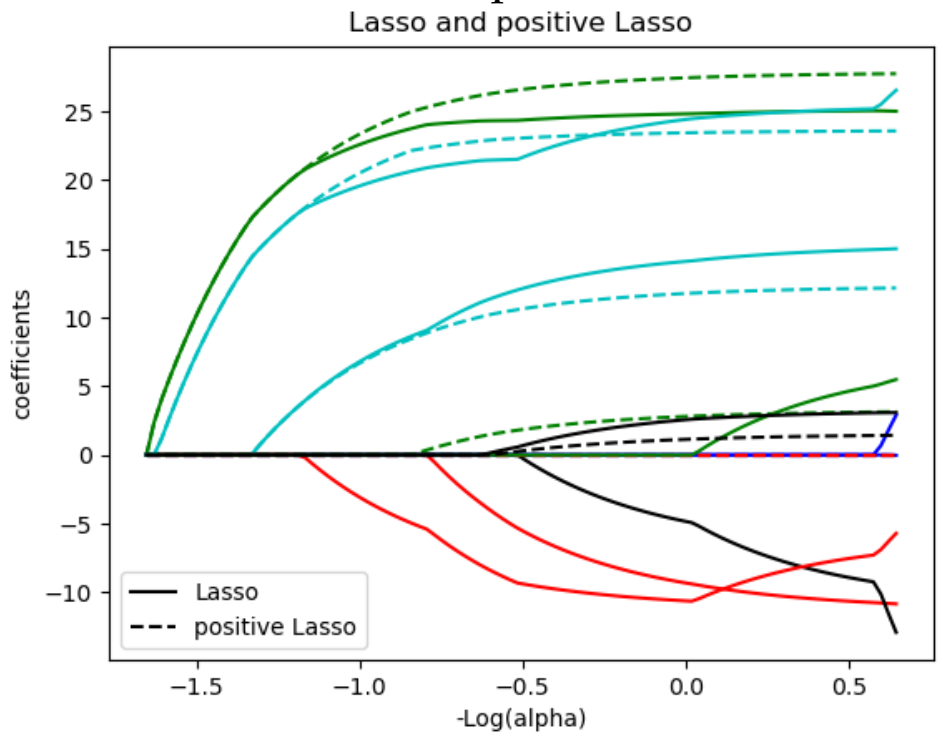
Think of an "equal cost" line H_0

Effect: linear regression example

L2



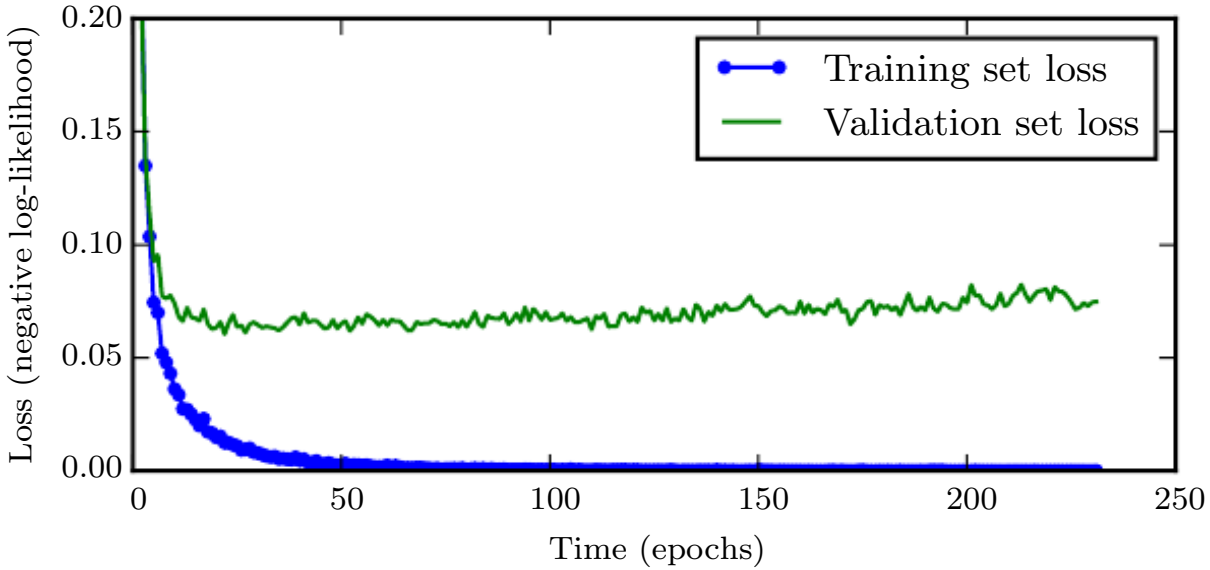
L1 and positive L1



Alpha == our lambda
https://scikit-learn.org/stable/modules/linear_model.html

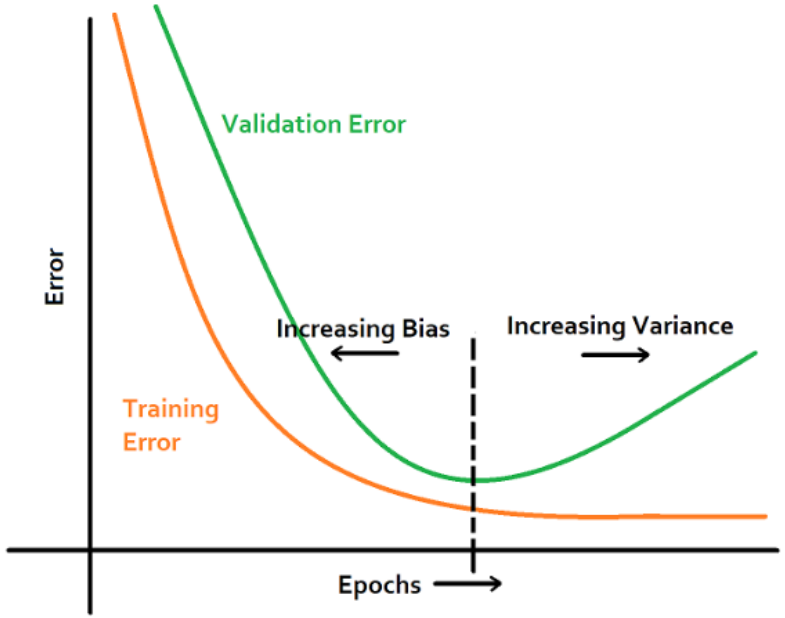
3) Early stopping

- Monitor performance on a separate validation set
- Training the network will decrease training error, as well validation error
- When validation error starts increasing, it is quite likely that the network starts to overfit.



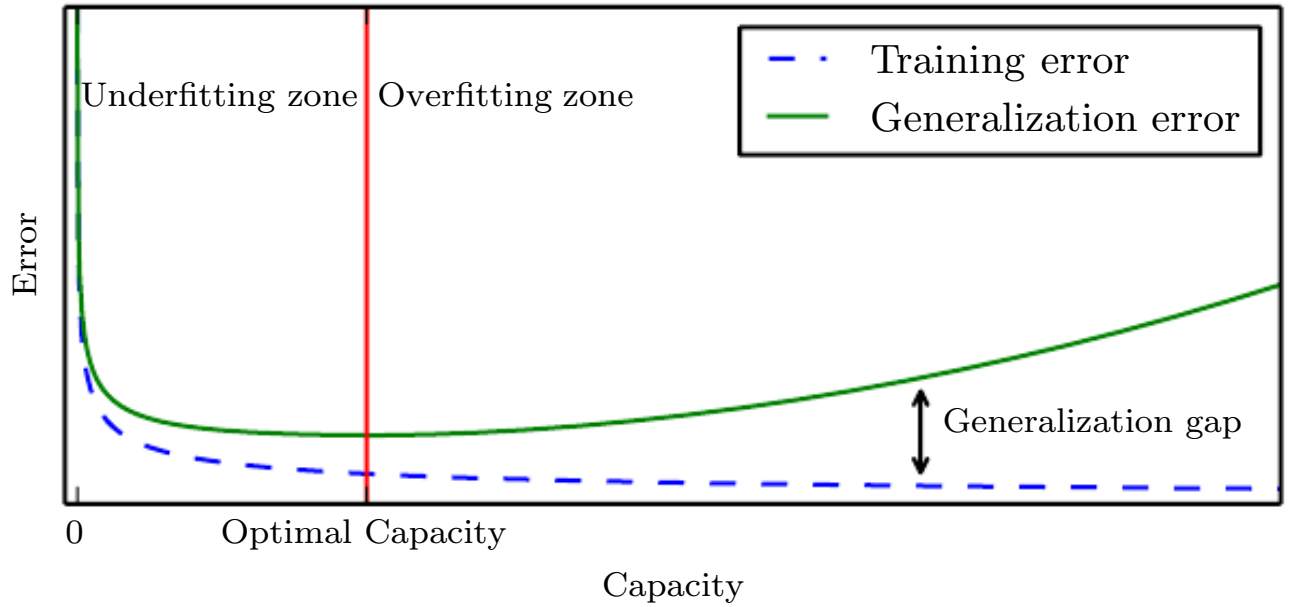
3) Early stopping

- We can obtain a model with better validation set error
  - The parameter setting at time point with the lowest validation set error.
  - The models at this stage have low variance and generalize the data well.
  - Further training would increase the variance of the model and lead to overfitting.
- This strategy is *Early Stopping*.
  - Commonly used, effectiveness and simplicity.
  - Need a validation set (different from test set!)
  - Need to maintain a copy of the best parameters.
  - Either alone or with other regularization strategies.



3) Early stopping

- We can think of early stopping is as an efficient hyperparameter selection:
  - The number of training steps is just another hyperparameter.
  - Most hyperparameters that control model capacity have such a U-shaped validation set performance curve.
  - When do we know when we're in the valley?



Why does early-stopping work as regularization?

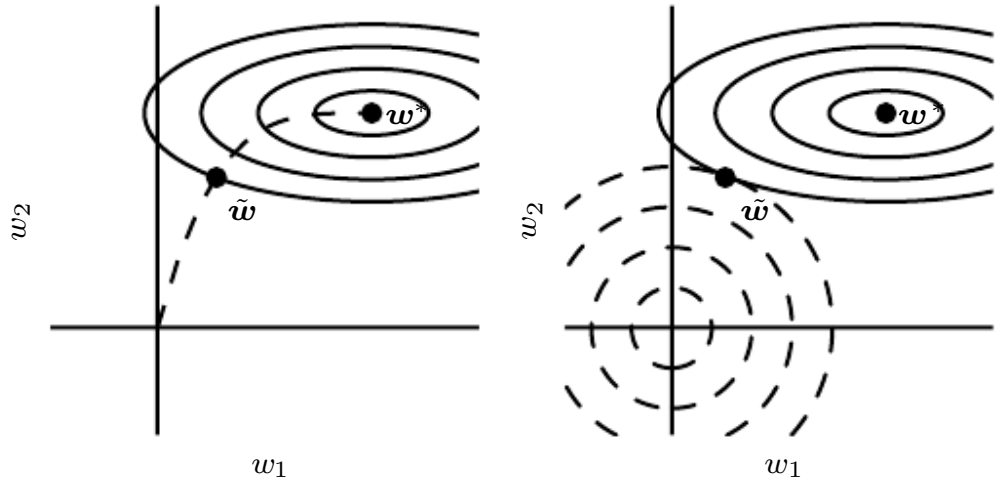
- For a *simple linear model* with a quadratic error function and simple gradient descent, early stoppixng is equivalent to ℓ_2-regularization.

$$\tau \approx \frac{1}{\epsilon\alpha}, \qquad \alpha \approx \frac{1}{\tau\epsilon}.$$

- α is the regularization constant, τ is no. of iterations, and ε is the learning rate.
- Increasing no. of epochs/iterations $\boldsymbol{\tau}$ is equivalent to reducing the regularization constant α.

Why does early-stopping work as regularization?

- Rather than stopping at the point $\boldsymbol{w}^*$ that minimizes the cost, early stopping results in the trajectory stopping at an earlier point $\tilde{\boldsymbol{w}}$.



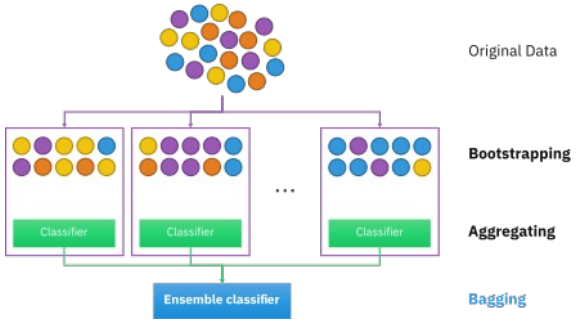
- Early stopping has the advantage over weight decay that early stopping automatically determines the correct "amount of regularization" while weight decay requires many training experiments with different values of its hyperparameter.

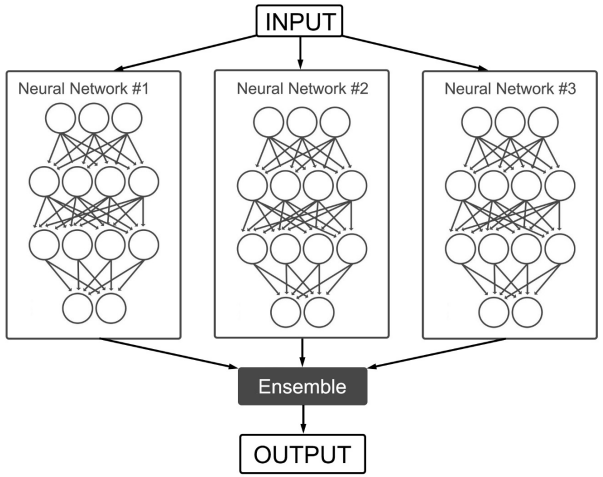
4) Dropout: the problem it addresses

- The co-adaptation phenomenon
  - Co-adaptation refers to when different hidden units in a neural networks have highly correlated behaviour.
  - It is common that some of the connections will have more predictive capability than the others.
  - These powerful connections are learned more while the weaker ones are ignored.
  - Over many iterations, only a fraction of the node connections is trained. And the rest stop participating.
- Dropout resolves this co-adaptation issue.

4) Dropout: why does it work?

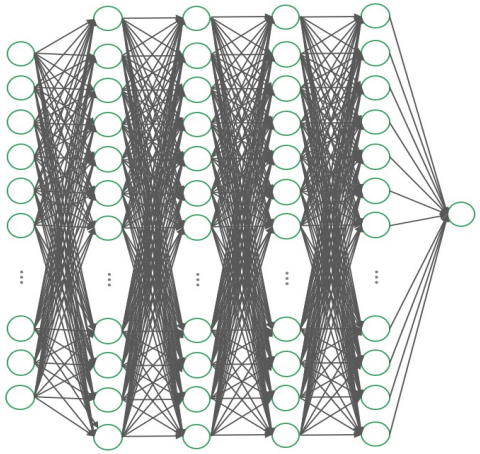
- Bagging
  - A technique for reducing generalization error by combining several models.
  - To train several different models separately, then have all of the models vote on the output for test examples.
  - Techniques employing this strategy are known as ensemble methods.
- Ensemble
  - Different kinds of models.
  - Different initializations.
  - Using a different algorithm.
  - Constructing different datasets.





4) Dropout: why does it work?

- Makes bagging practical for ensembles of very many neural nets.
  - Bagging seems impractical for a large neural network.
  - Training and evaluating is costly in terms of runtime and memory.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

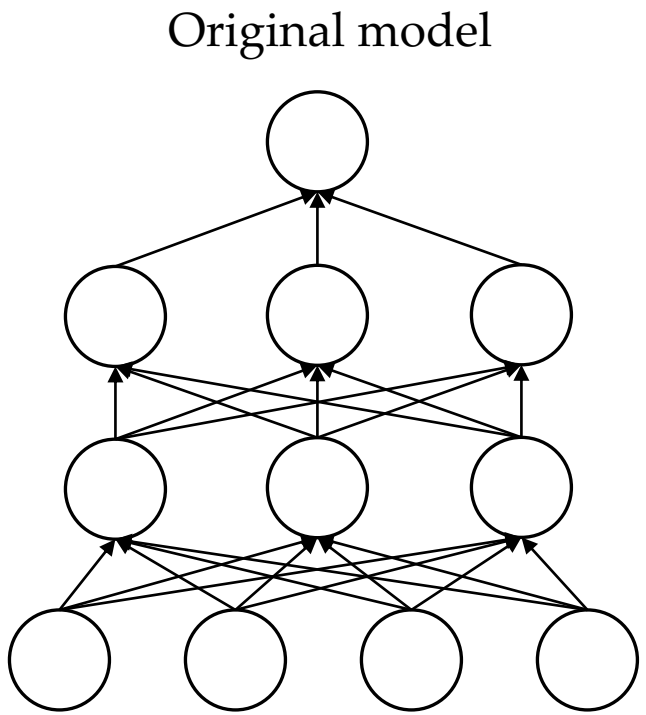


4) Dropout: how is it implemented?

- During training randomly set activations to 0
  - Neurons sampled at random from a Bernoulli distribution with p (eg, $p = 0.5$)
  - Neuron activations reweighted by $1/p$
- During testing all neurons are used
- Benefits
  - Reduces complex co-adaptations between neurons
  - Every neuron becomes more robust
  - Decreases overfitting
- Not super common for large scale datasets.
  However in different form more common: “DropPath/stoch. Depth”

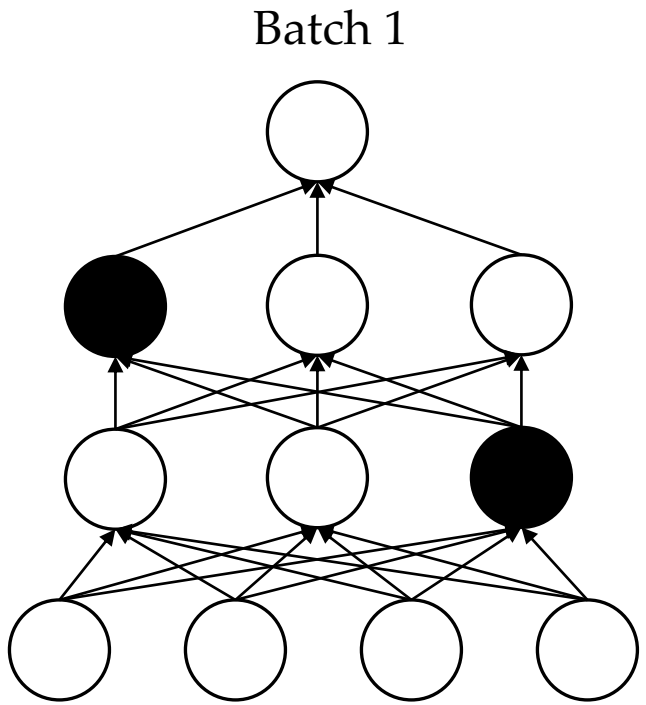
Dropout

- Effectively, a different architecture for every input batch during training



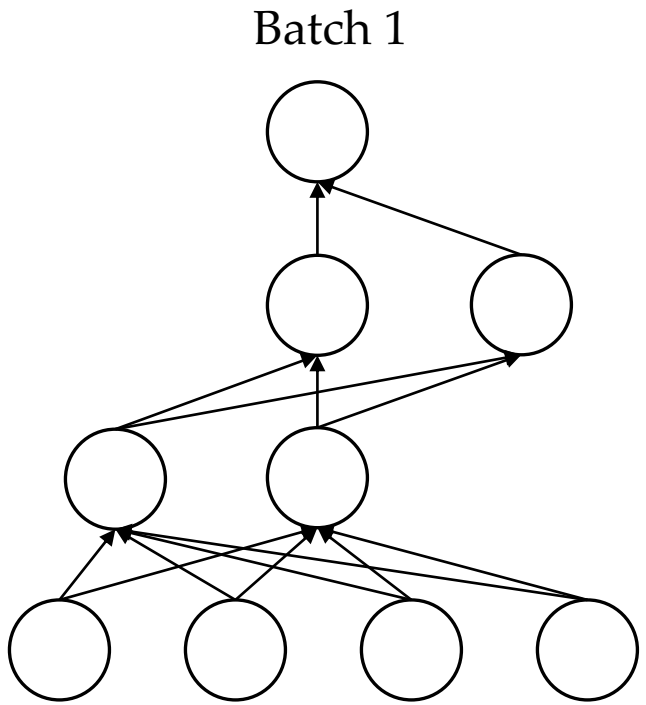
Dropout

- Effectively, a different architecture for every input batch during training



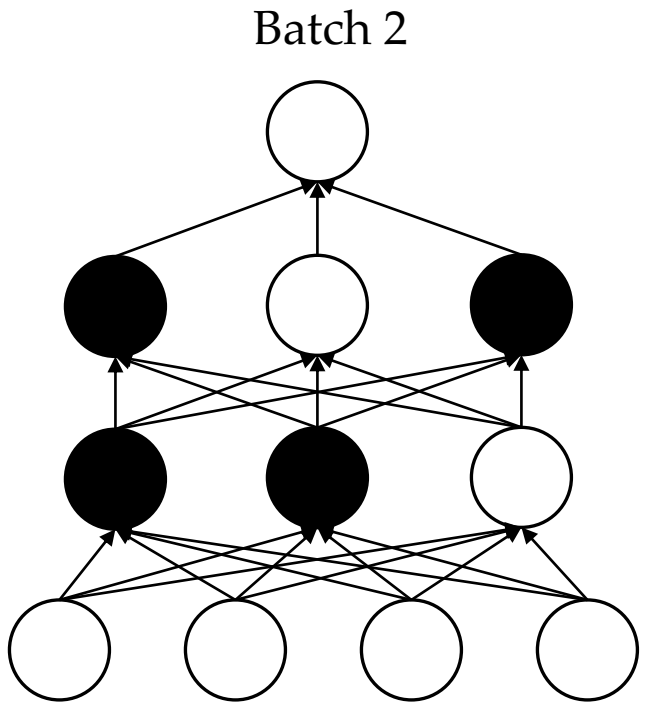
Dropout

- Effectively, a different architecture for every input batch during training



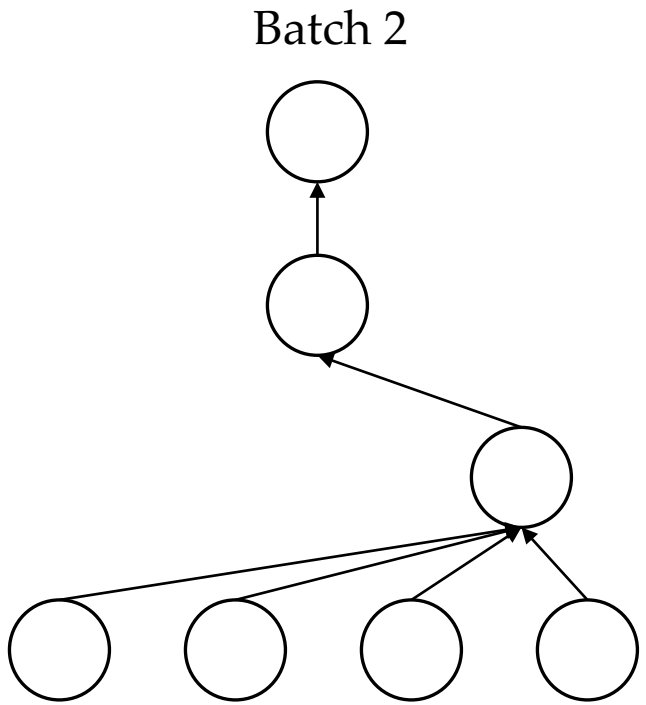
Dropout

- Effectively, a different architecture for every input batch during training



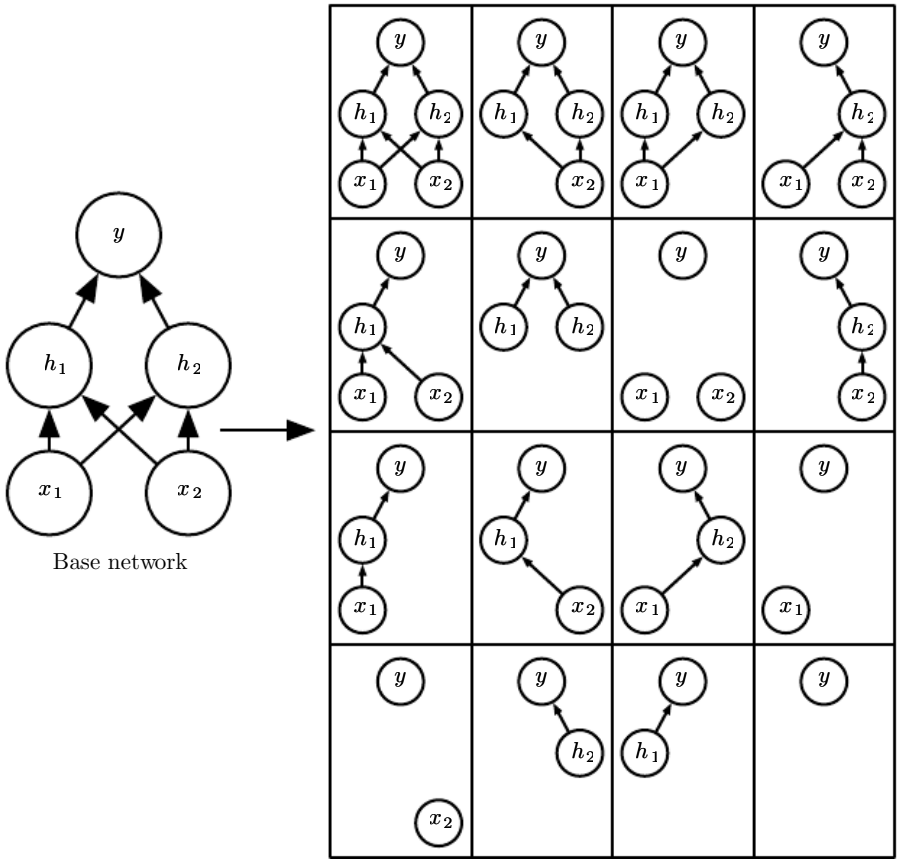
Dropout

- Effectively, a different architecture for every input batch during training



Dropout

- Effectively, a different architecture for every input batch during training



Dropout *vs.* Bagging

- Bagging
  - The models are all independent
  - Each model is trained to convergence on its respective training set.
- Dropout
  - The "models" share parameters, with each model inheriting a different subset of parameters from the parent neural network.
  - Typically, most models are not explicitly trained at all.
  - The training set encountered by each sub-network is indeed a subset of the original training set sampled with replacement.

Dropout beyond Bagging

- Dropout trains not just a bagged ensemble of models, but an ensemble of models that share hidden units.
- Each hidden unit must be able to perform well regardless of which other hidden units are in the model.
- Hidden units must be prepared to be swapped and interchanged between models.
- *Dropout thus regularizes each hidden unit to be not merely a good feature but a feature that is good in many contexts.*

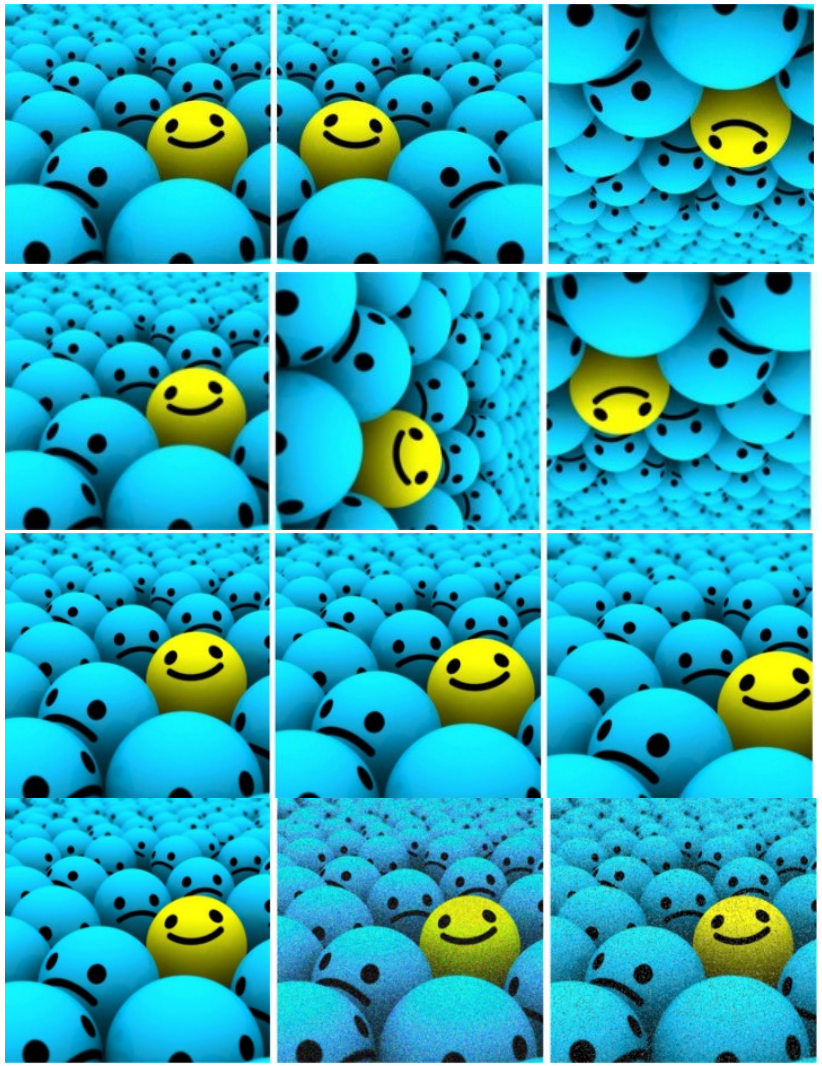
5) Data augmentation

- *The best way to make a machine learning model generalize better is to train it on more data.* (see: "The unreasonable effectiveness of data")
  - Data* is limited in practice
  - One way is to create fake data – *Data Augmentation***
- *Your neural network is only as good as the data you feed it.*
- By performing augmentation, we can prevent neural networks from learning or memorizing irrelevant patterns, essentially boosting overall performance.

* Labeled data
** Not that trivial. Augmentations are more than just fake data. See lecture 13 on self-supervised learning

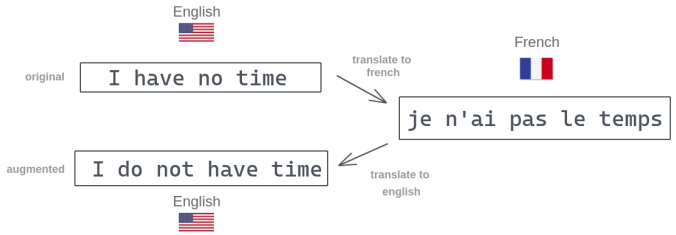
Data augmentation

- Augmentation techniques in vision
  - Flip
  - Rotation
  - Scale
  - Crop
  - Translation
  - Gaussian noise
- Be aware of label change
  - “b” and “d”
  - “6” and “9”



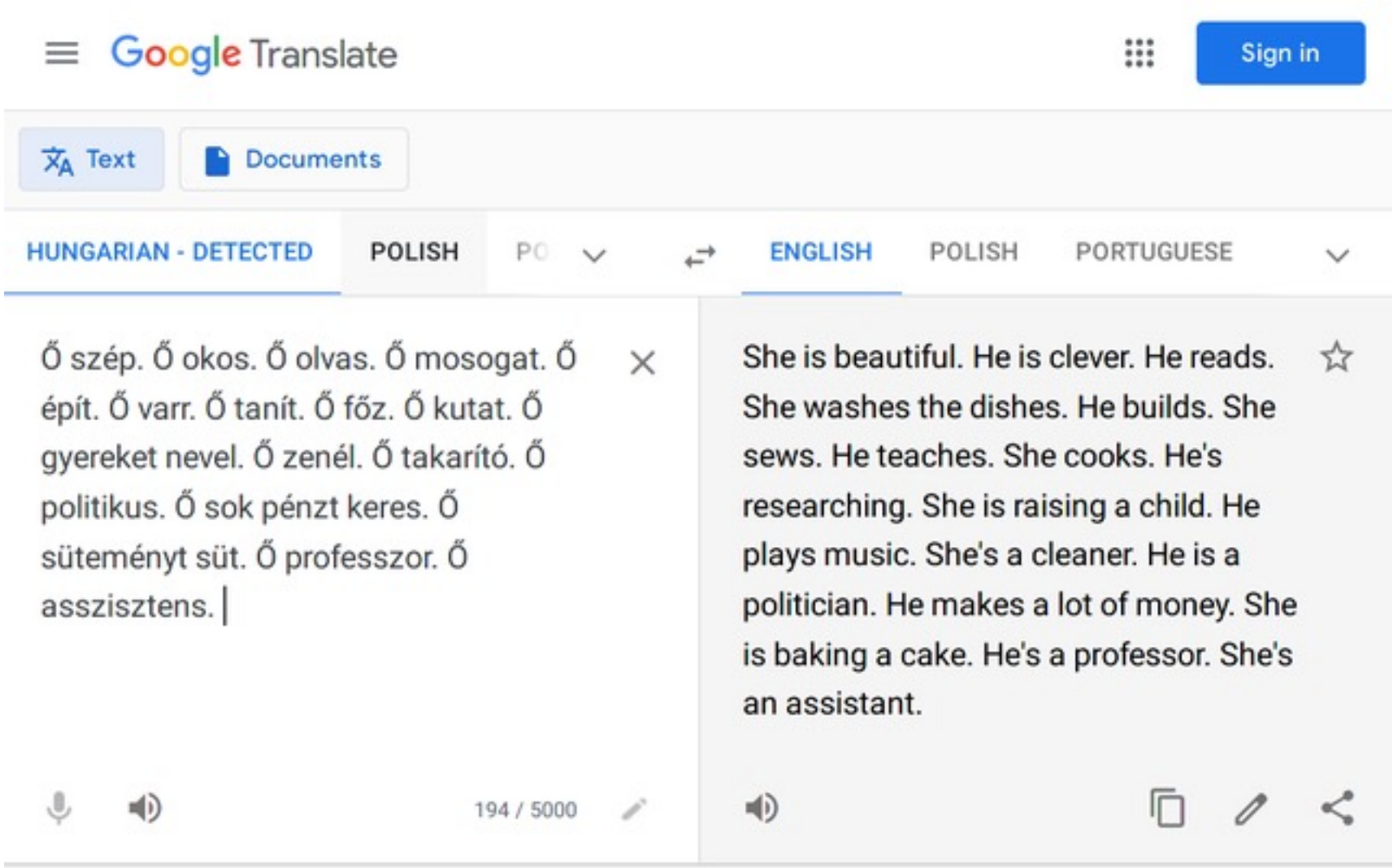
Link

- In NLP
  - Backtranslation

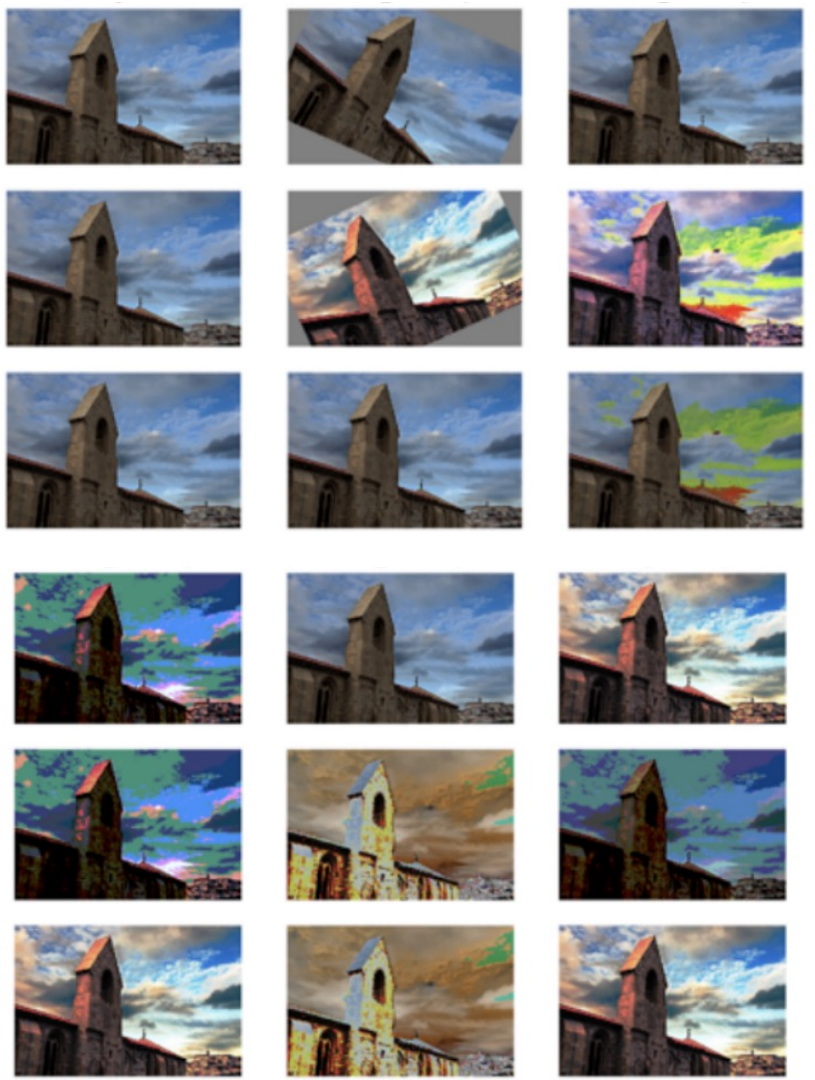


  - Synonym replacement
  - Random insertion
  - Random deletion
  - Random swapping

One note about backtranslation though:



Common computer vision augmentations visualised



Data augmentation

- Is effectively another hyperparameter, ie you cannot run SGD on it to find best augmentations.
  - But Reinforcement Learning: AutoAugment, RandAugment etc.
- Hard to tune & domain dependent:
  - Eg, What Should Not Be Contrastive in Contrastive Learning. Xiao et al. ICLR 2021: if you want to identify specific flower types… randomly changing the color as augmentation is not so good.
- One interpretation: a way to incorporate "domain knowledge" into the network. How else could we say that all these below are "same"?

Data augmentation

- It really works. Major part of state of the art training pipelines

Method	Depth	Params	C10	C10+	C100	C100+
Network in Network [22]	-	-	10.41	8.81	35.68	-
All-CNN [32]	-	-	9.08	7.25	-	33.71
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57
Highway Network [34]	-	-	-	7.72	-	32.39
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73
ResNet [11]	110	1.7M	-	6.61	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58
	1202	10.2M	-	4.91	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07
	28	36.5M	-	4.17	-	20.50
with Dropout	16	2.7M	-	-	-	-
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33
	1001	10.2M	10.56*	4.62	33.47*	22.71
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**

Table 2: Ingredients and hyper-parameters used for ResNet-50 training in different papers. We compare existing training procedures with ours.

	Previous approaches					Ours		
Procedure → Reference	ResNet [13]	PyTorch [1]	FixRes [48]	DeiT [45]	FAMS (×4) [10]	A1	A2	A3
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	-	$0.02^{t/400}$	-	-	-
decay epochs	30	30	30	-	1	-	-	-
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	0.1	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

Other regularizations

- Noise robustness
  - adding noise to weights - uncertainty
  - adding noise to outputs - label smoothing
- Semi or self-supervised learning
  - introducing a particular form of prior belief about the solution
  - smoothness, generaliseability
- Multi-task learning
  - shared the input and parameters – improve the statistical strength
  - require statistical relationship between tasks

Various ways to regularise

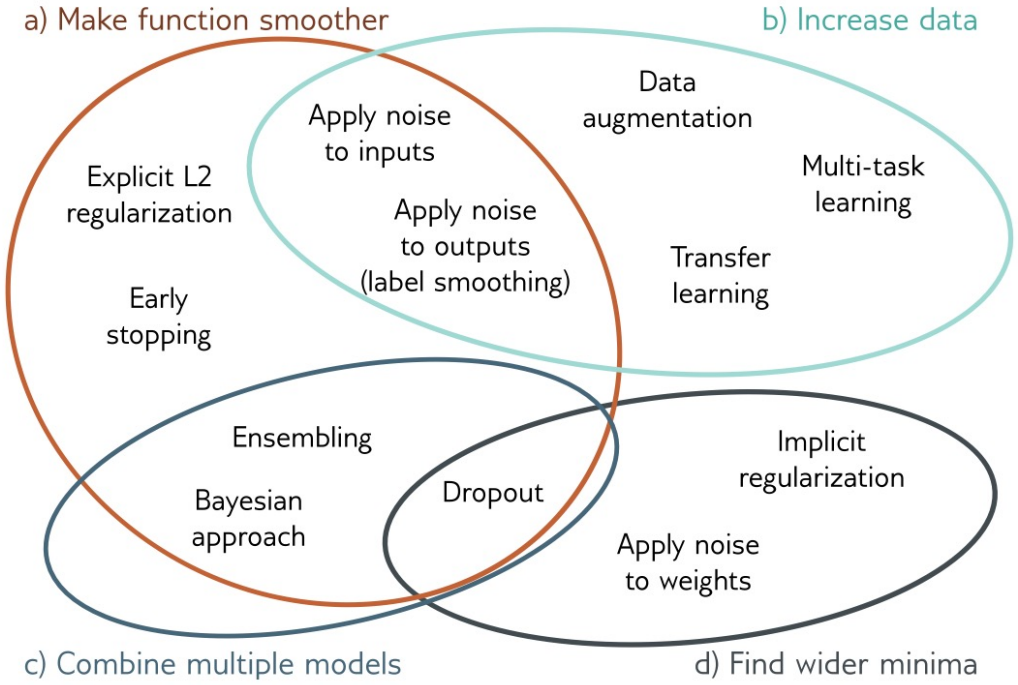


Figure 9.14 Regularization methods. The regularization methods discussed in this chapter aim to improve generalization by one of four mechanisms. a) Some methods aim to make the modeled function smoother. b) Other methods increase the effective amount of data. c) The third group of methods combine multiple models and hence mitigate against uncertainty in the fitting process. d) Finally, the fourth group of methods encourages the training process to converge to a wide minimum where small errors in the estimated parameters are less important.

- UDL book, Chapter 9

Normalization

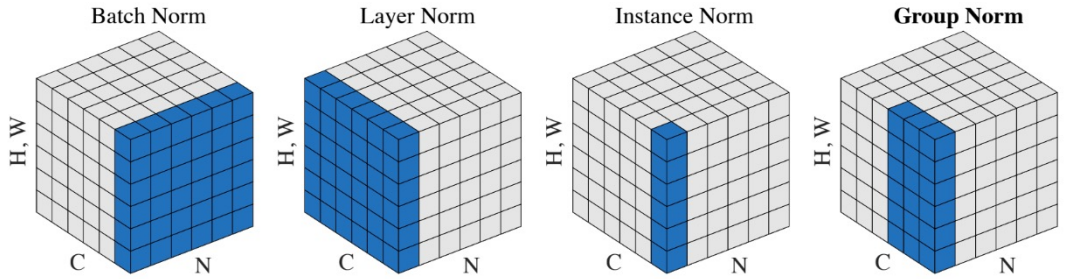
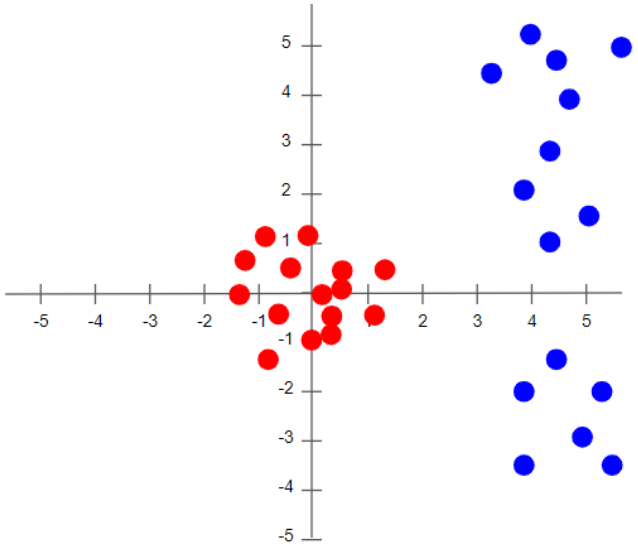
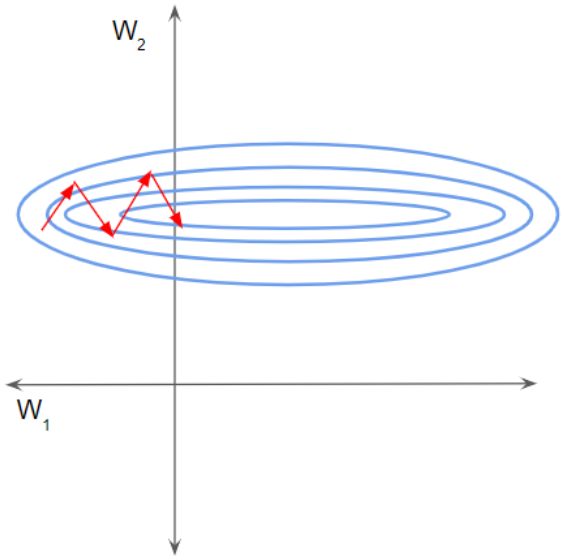


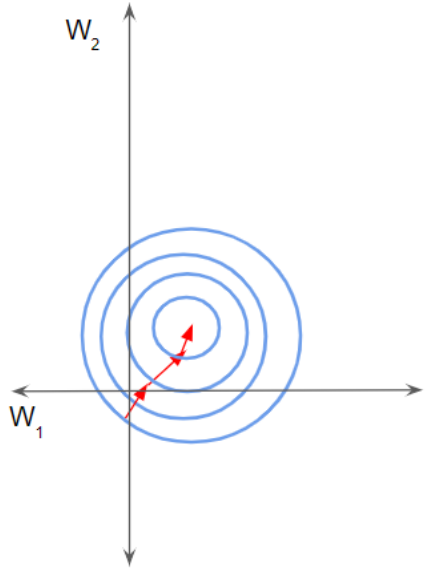
Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Data preprocessing

- Normalization
  - The data pre-processing to bring the numerical data to a common scale without distorting its shape.
  - The reason is partly to ensure that our model can generalize appropriately.
- This ensures that all the feature values are now on the same scale.





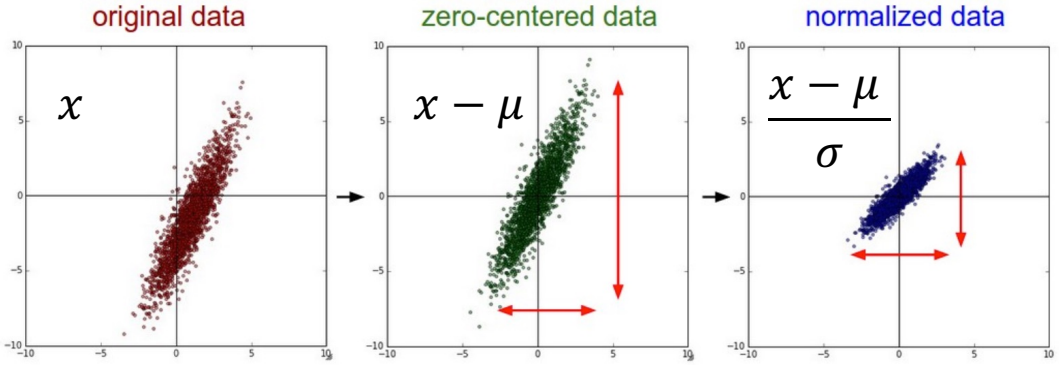


Link

Normalizing Input Data

- Transforming the input to zero-mean, unit variance
  - Assume: Input variables follow a Gaussian distribution (roughly)
  - Subtract input by the mean
  - Optionally, divide by the standard deviation

$$N(\mu, \sigma^2) \rightarrow N(0, 1)$$

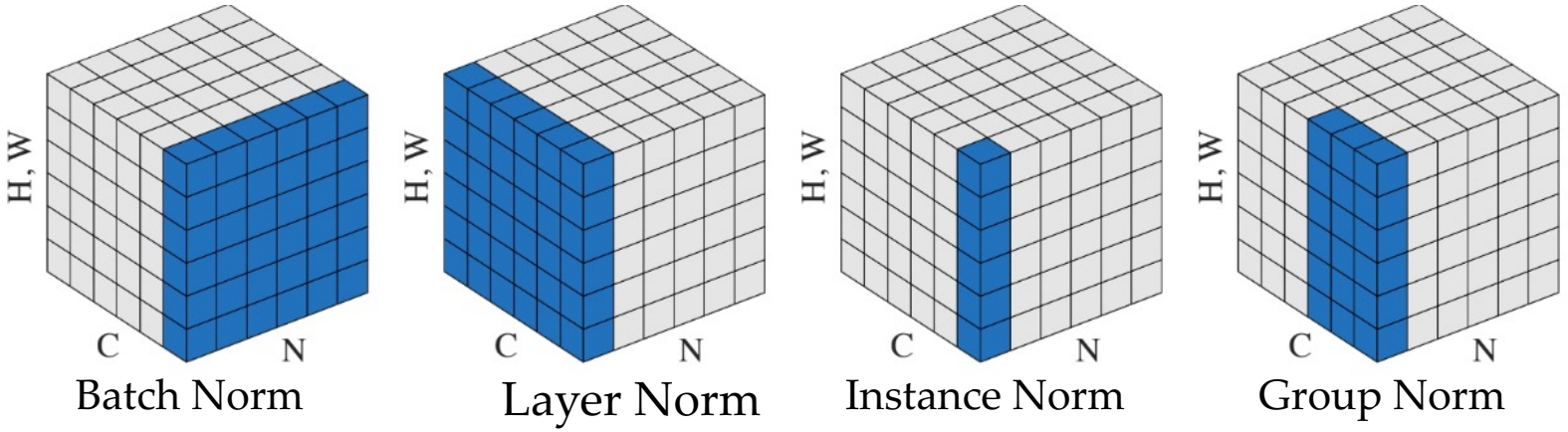


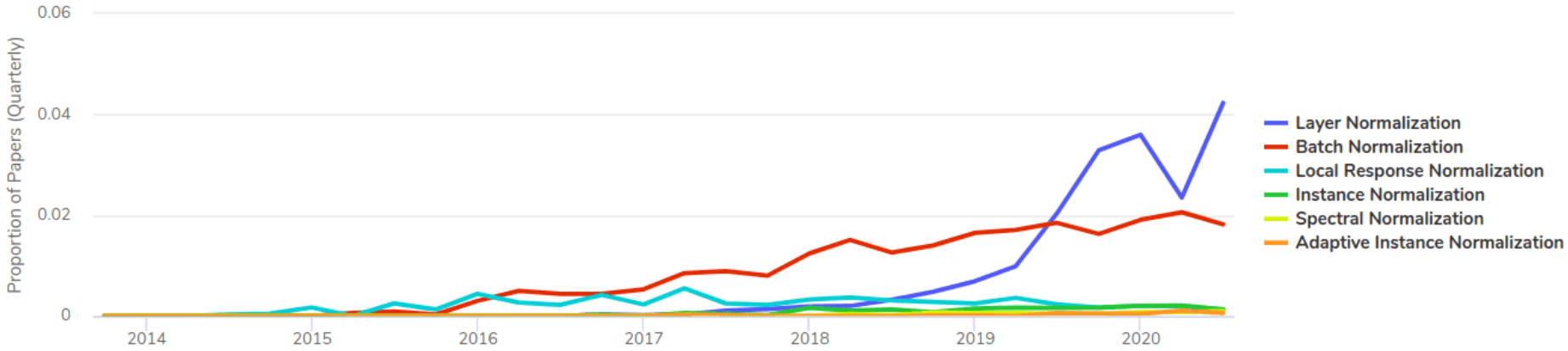
Picture credit: Stanford Course

Eg ImageNet: mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225]

Normalizing intermediate layers

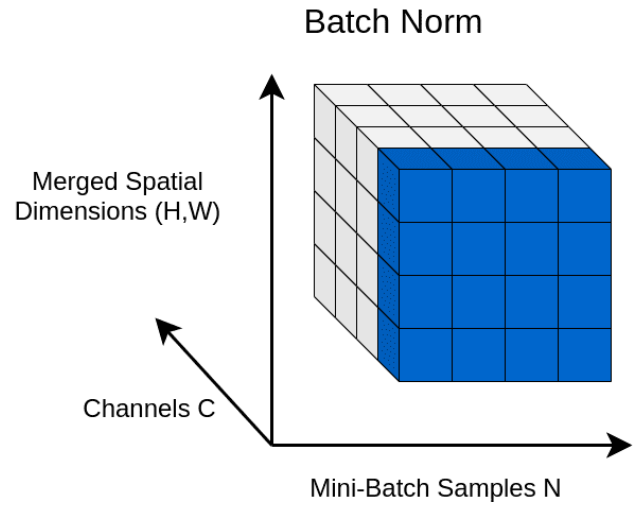
- Batch normalization
- Layer normalization
- Instance normalization
- Group normalization
- Weight normalization





Batch normalization

- The activations from the previous layer are simply the inputs to this layer.
- Batch normalization is a process to make neural networks faster and more stable through adding extra layers in a deep neural network.
- Takes place in batches, not as a single input.
- This normalization is applied typically before activation.

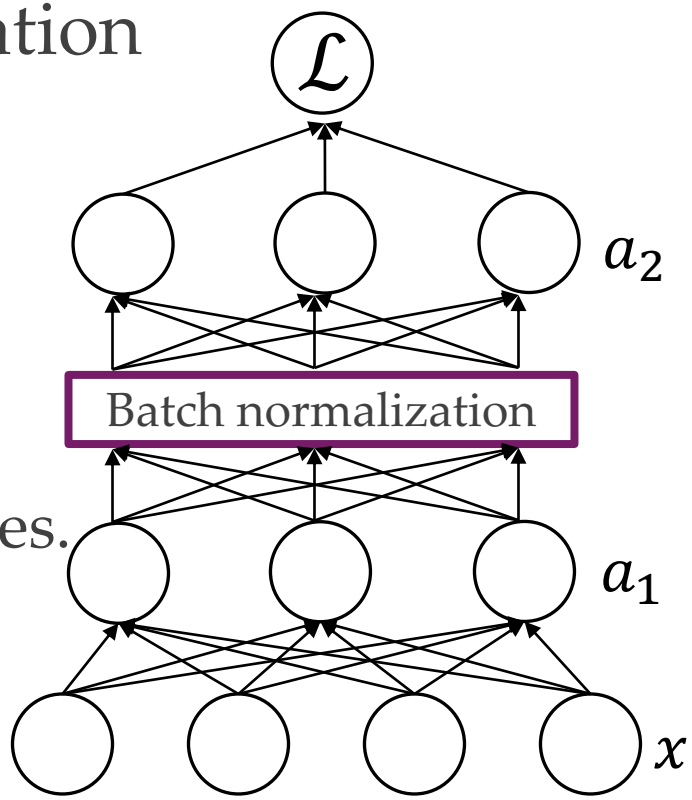


Batch normalization

- Normalize the layer inputs with batch normalization
  - Normalize $a_l \sim N(0, 1)$
  - Followed by affine transformation

$$a_l \leftarrow \gamma a_l + \beta$$

- The parameters γ and β are trainable
- Used for re-scaling (γ) and shifting (β) of the vector values.
- Ensure the optimal values of γ and β are used.
- Enable the accurate normalization of each batch.



Batch normalization – The algorithm

- i runs over mini-batch samples, j over the feature dimensions

$\mu_j \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_{ij}$ [compute mini-batch mean]

$\sigma_j^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_{ij} - \mu_j\right)^2$ [compute mini-batch variance]

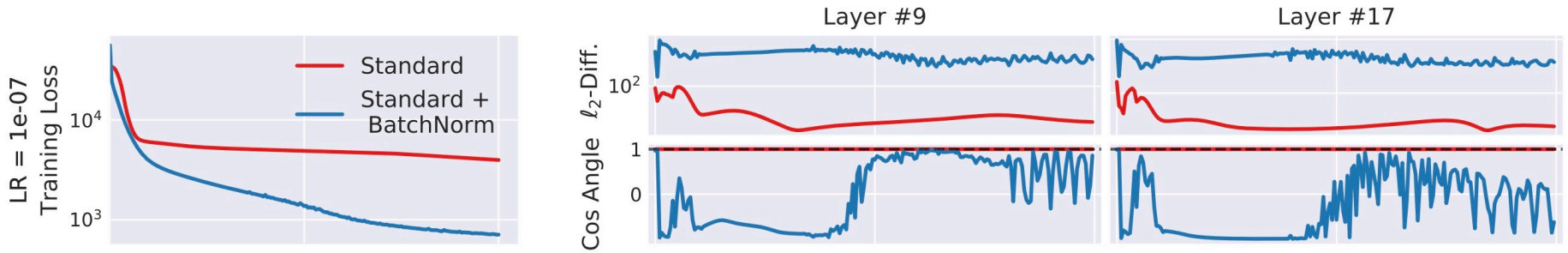
$\hat{x}_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\varepsilon}}$ [normalize input]

$\hat{x}_{ij} \leftarrow \gamma\hat{x}_{ij} + \beta$ [scale and shift input]

Trainable parameters

How does batch normalization help optimization?

- Internal covariate shift ?
  - The change in the distribution of layer inputs caused by updates to the preceding layers.
  - Little concrete evidence supporting it.
  - Seems no link between the performance gain of BN and the reduction of internal covariate shift.

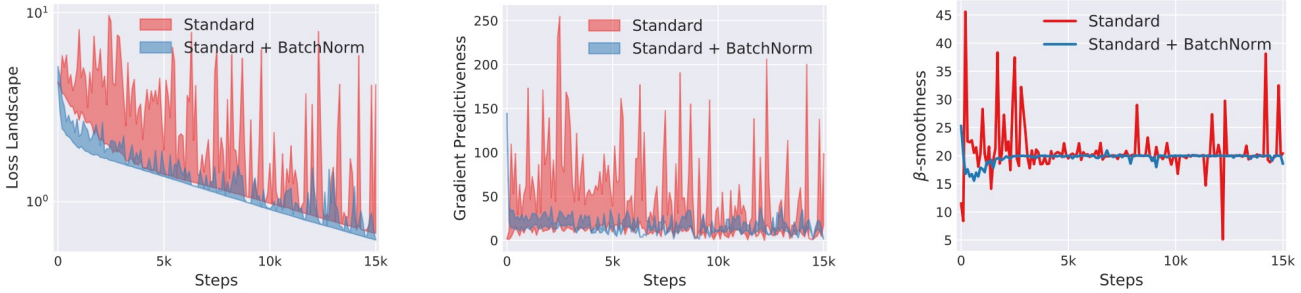


Deep linear networks (25 layers)

Santurkar, et al., How Does Batch Normalization Help Optimization?, NeurIPS, 2018

How does batch normalization help optimization?

- The impact of BatchNorm on the stability of the loss
  - It makes loss landscape significantly smoother.
  - Improvement in the Lipschitzness of the loss function.
  - It makes *gradients* of the loss more Lipschitz too.
  - it makes the gradients more reliable and predictive.



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

- Recently: also networks without normalisation, but careful initialisation *[High-Performance Large-Scale Image Recognition Without Normalization. Brock et al. 2021]*

Santurkar, et al., How Does Batch Normalization Help Optimization?, NeurIPS, 2018

Benefits of Batch normalization

- Networks train faster
- Allows higher learning rates
- Makes weights easier to initialize
- Makes more activation functions viable
- The added noise reduces overfitting
- Simplifies the creation of deeper networks
- Provides a bit of regularization
- May give better results overall

Quiz:

How would you use batch normalization at test time?

A) go through the dataset in a predefined order
B) go through the dataset in a random order
C) go through the dataset in a random order multiple times
D) simply skip it

Batch normalization at test time

- How do we ship the Batch Norm layer after training?
  - We might not have batches at test time
  - Batches are random? -> not reproducible

- Usually: keep a moving average of the mean and variance during training
  - Plug them in at test time
  - To the limit, the moving average of mini-batch statistics approaches the batch statistics

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

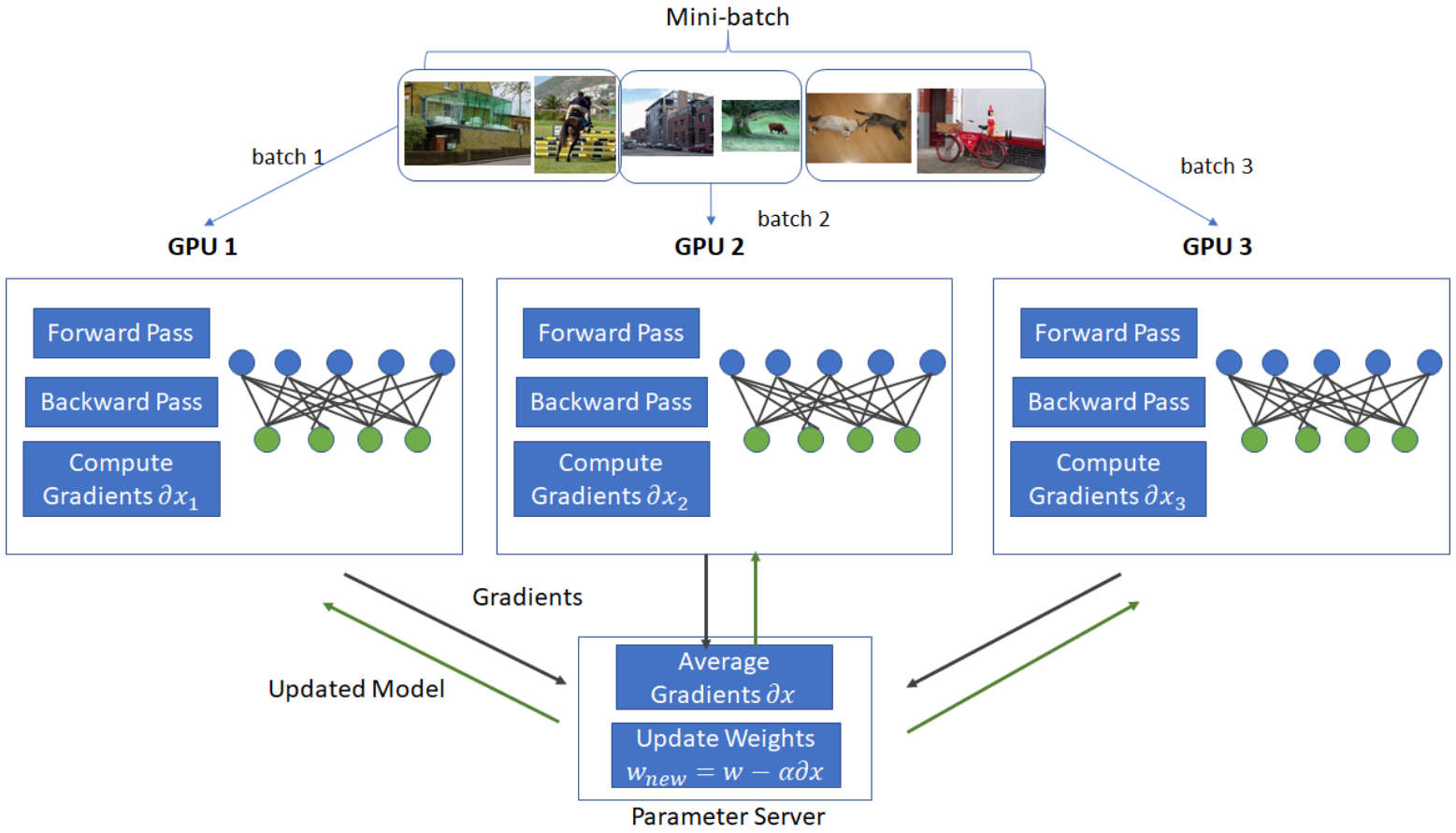
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

$$\hat{y}_i \leftarrow \gamma \hat{x}_i + \beta$$

Disadvantages of batch normalization

- Requires large mini-batches
  - Cannot work with mini-batch of size 1 ($\sigma = 0$)
  - Performance is sensitive to the batch size
  - Very memory intense, all the batch statistics must be stored in the layer.
  - Discrepancy between training and test data
  - Breaks the independence between training examples in the minibatch
  - Not applicable to online learning
  - Can limit model capability (zero-mean)
- Awkward to use with recurrent neural networks
  - Must interleave it between recurrent layers
  - Also, store statistics per time step
- Often the reason for bugs
- There are more reasons.

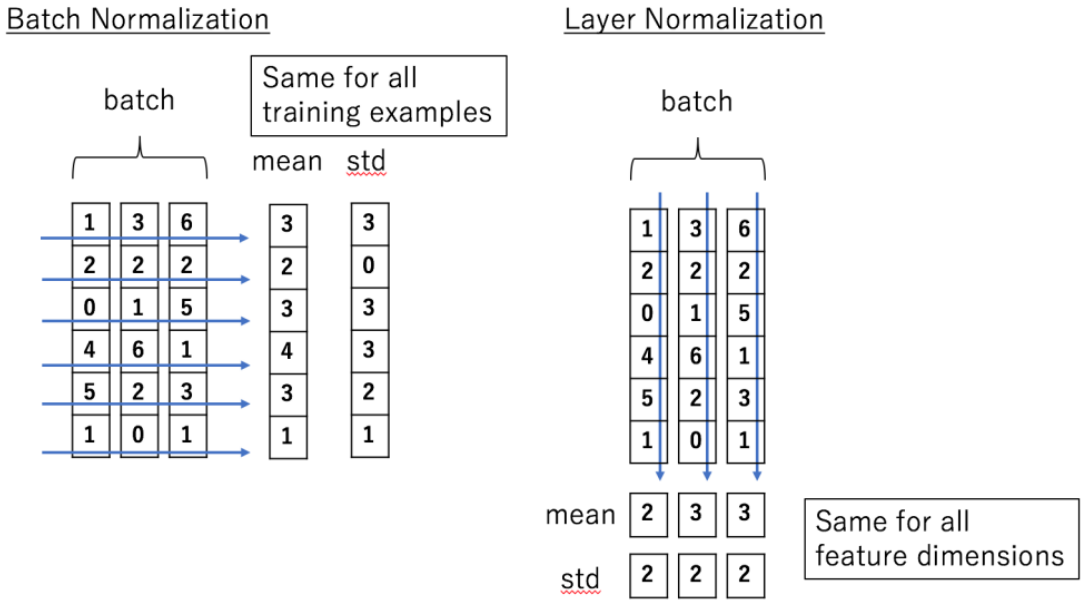
Disadvantages of batch normalization with distributed training

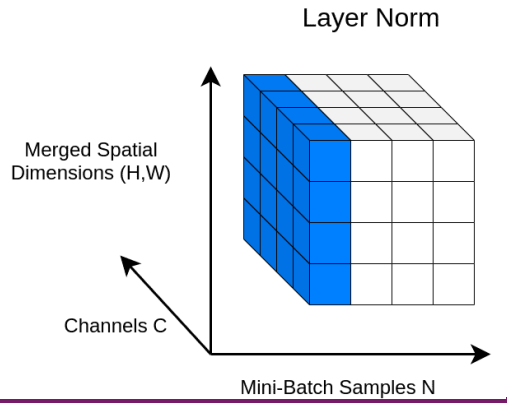


Generally: using multiple GPUs for training a model is very common.

Layer normalization

- The statistics (mean and variance) are computed across all channels and spatial dimensions.
- The statistics are independent of the batch.
- This layer was initially introduced to handle vectors (mostly the RNN outputs).
- Layer normalization performs exactly the same computation at training and test times





Layer Normalization, Ba, Kiros, Hinton, 2016

Layer normalization (LN)

- i runs over mini-batch samples, j over the feature dimensions

$\mu_i \leftarrow \frac{1}{m}\sum_{j=1}^{m} x_{ij}$ [mean over features]

$\sigma_i^2 \leftarrow \frac{1}{m}\sum_{j=1}^{m}(x_i - \mu_B)^2$ [variance over features]

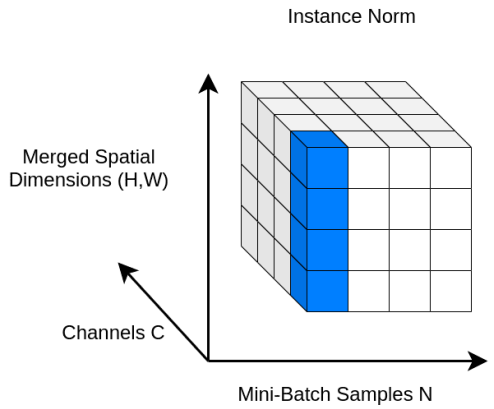
$\widehat{x_i} \leftarrow \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$ [normalize input]

$\widehat{y_i} \leftarrow \gamma\widehat{x_i} + \beta$ [scale and shift input]

Layer Normalization, Ba, Kiros, Hinton, 2016

Instance normalization (IN)

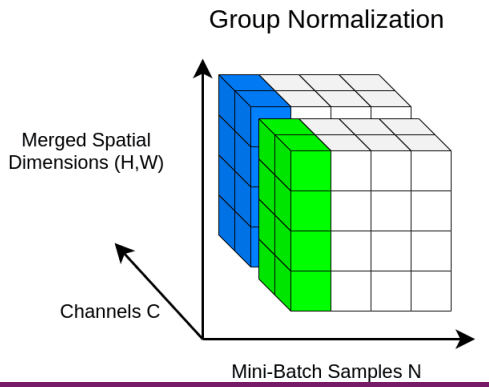
- Similar to layer normalization but per channel per training example.
- Basic idea: network should be agnostic to the contrast of the original image.
- The statistics are computed only across the features' spatial dimensions.
- Literally, we just remove the sum samples compared to BatchNorm
- Originally proposed for style transfer
  - Not as good in image classification



Instance Normalization: The Missing Ingredient for Fast Stylization, Ulyanov, Vedaldi, Lempitsky, 2017

Group normalization (GN)

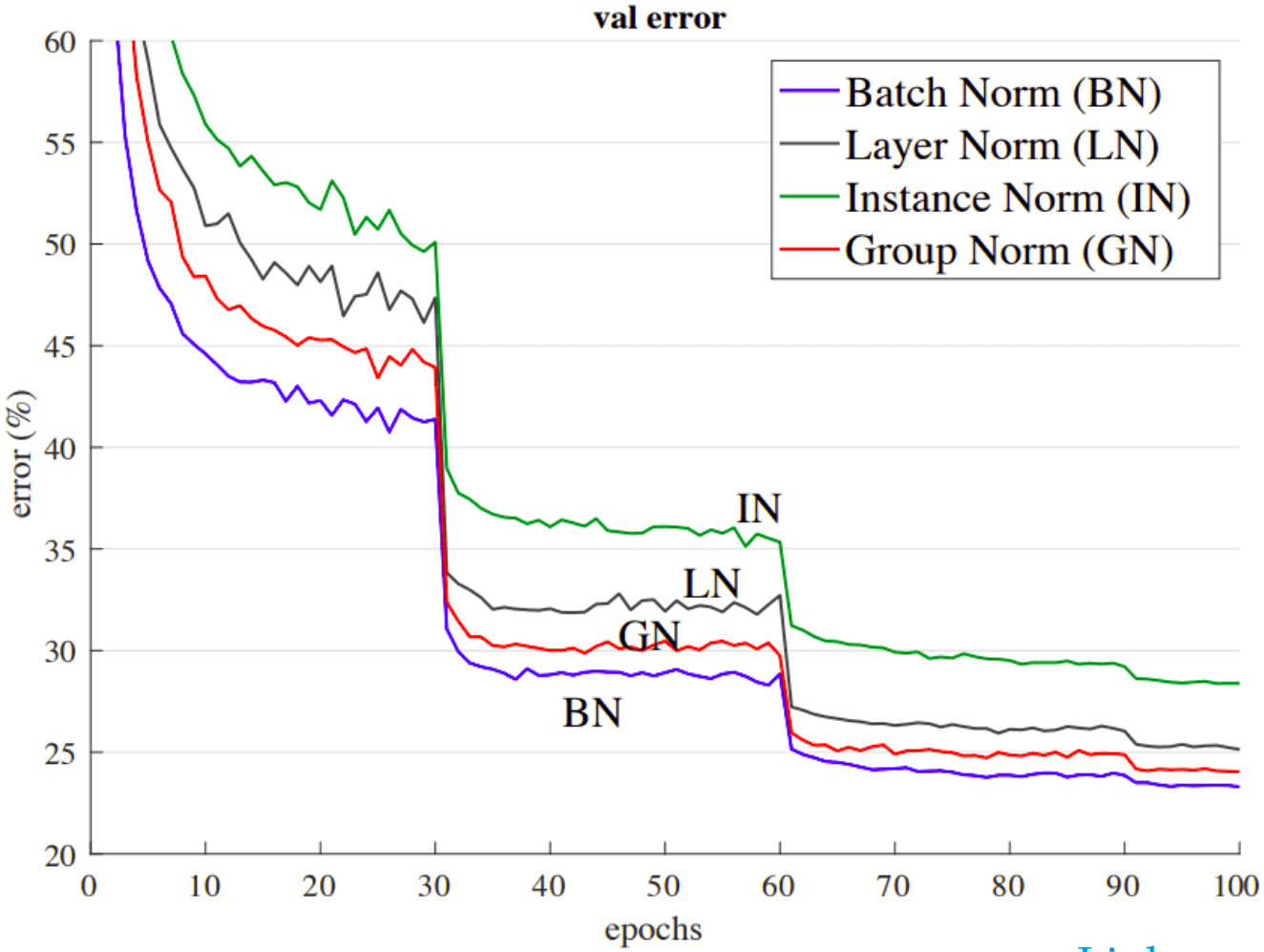
- Divides the channels into groups and computes statistics within each group.
- Same as instance norm but over groups of channels
  - Between layer normalization and instance normalization
  - # group = # channels -> instance norm
  - # group = 1 -> layer norm
- Better than batch normalization for small batches (e.g., <32)
  - Competitive for larger batches
- Useful for object detection/segmentation networks
  - rely on high resolution images
  - cannot have big mini-batches
- **Grouped convs** useful for larger networks, c.f. ResNeXt



Group Normalization, We, He, 2018

A comparison of different normalizations

- RestNet-50
- Batch size = 32
- But with some fixes, LayerNorm performs on par/better.



Link

Weight normalization

- Instead of normalizing activations, normalize weights
- The idea is to decouple the length from the direction of the weight vector and hence reparameterize the network to speed up the training.
- Re-parameterize weights

$$\boldsymbol{w} = g\frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$$

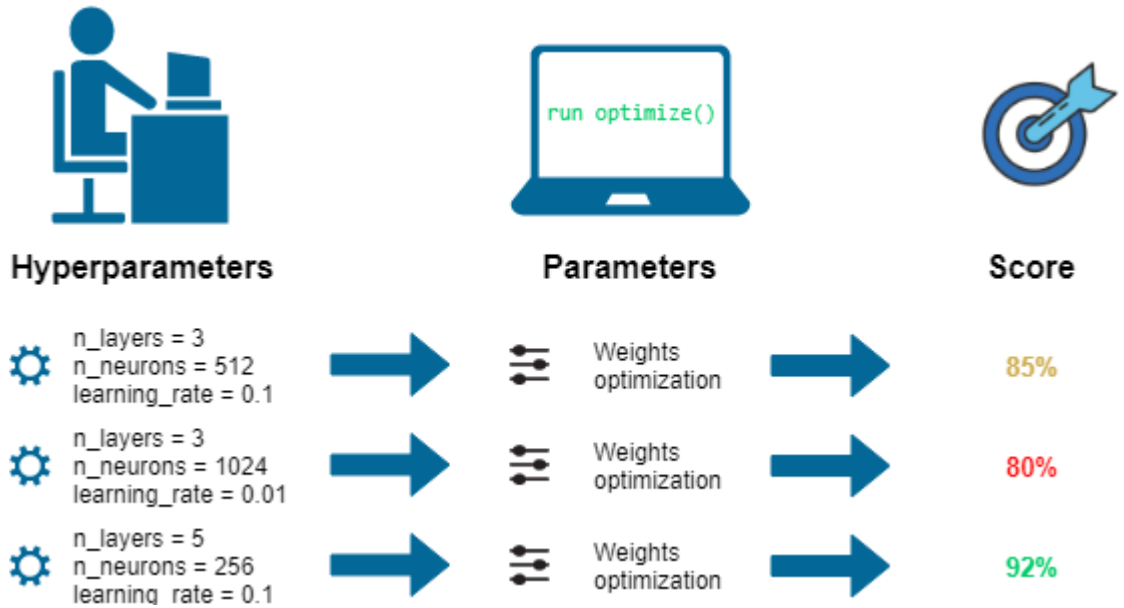
- Similar to dividing by standard deviation in batch normalization
- Can be combined with mean-only batch normalization
  - Subtract the mean (but not divide by the standard deviation)
  - Then, apply weight normalization

Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, Salimans, Kingma, 2016

References

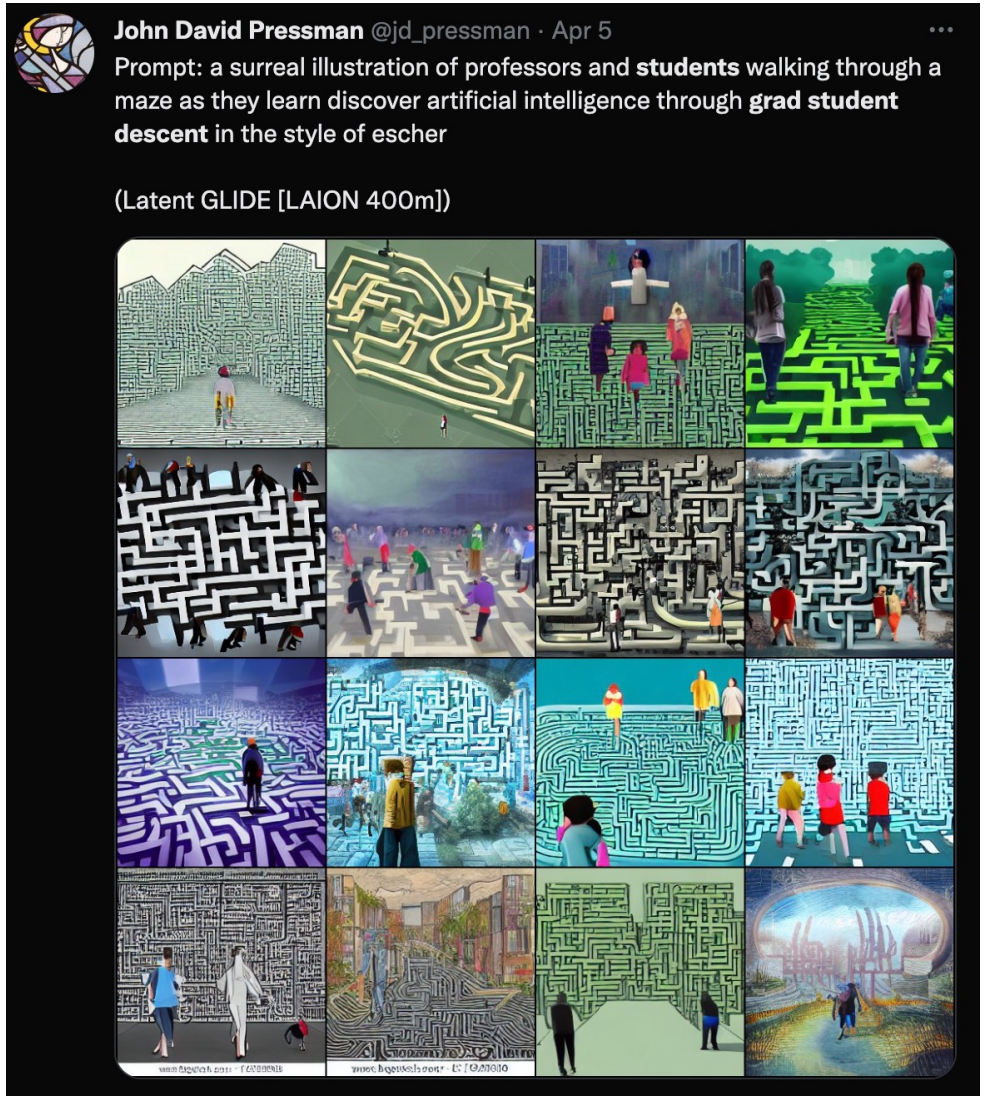
- Ioffe, S., & Szegedy, C. (2015).Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.
- Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Advances in neural information processing systems (pp. 901-909).
- Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.
- Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022.
- Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).
- Zhang, H., Dana, K., Shi, J., Zhang, Z., Wang, X., Tyagi, A., & Agrawal, A. (2018). Context encoding for semantic segmentation. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 7151-7160).
- Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In Advances in Neural Information Processing Systems (pp. 2483-2493).
- Dumoulin, V., Shlens, J., & Kudlur, M. (2016). A learned representation for artistic style. arXiv preprint arXiv:1610.07629.
- Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2337-2346).
- Huang, X., & Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1501-1510).
- Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2019). Big transfer (BiT): General visual representation learning. arXiv preprint arXiv:1912.11370.
- Qiao, S., Wang, H., Liu, C., Shen, W., & Yuille, A. (2019). Weight standardization. arXiv preprint arXiv:1903.10520.

Hyperparameters



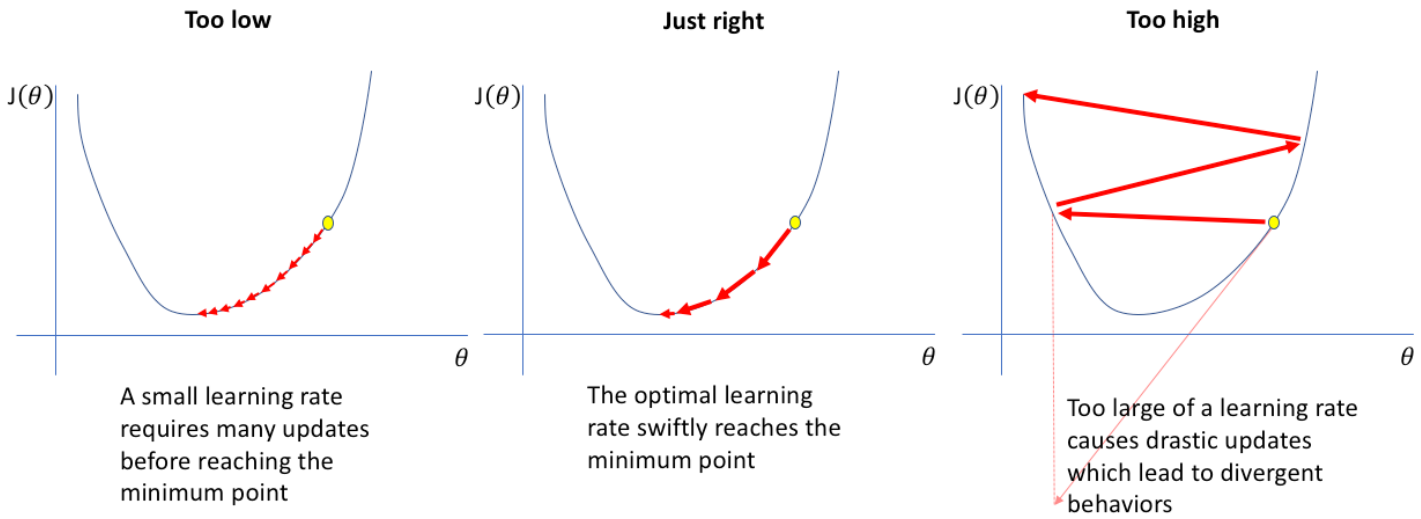
Hyperparameters
-
the stuff that cannot be tuned with SGD

but you can run grad student descent!
(though seems tricky ->



Learning rate

- The right learning rate η_t very important for fast convergence
  - Too strong → gradients overshoot and bounce
  - Too weak → slow training
- Learning rate per weight is often advantageous
  - Some weights are near convergence, others not



Convergence

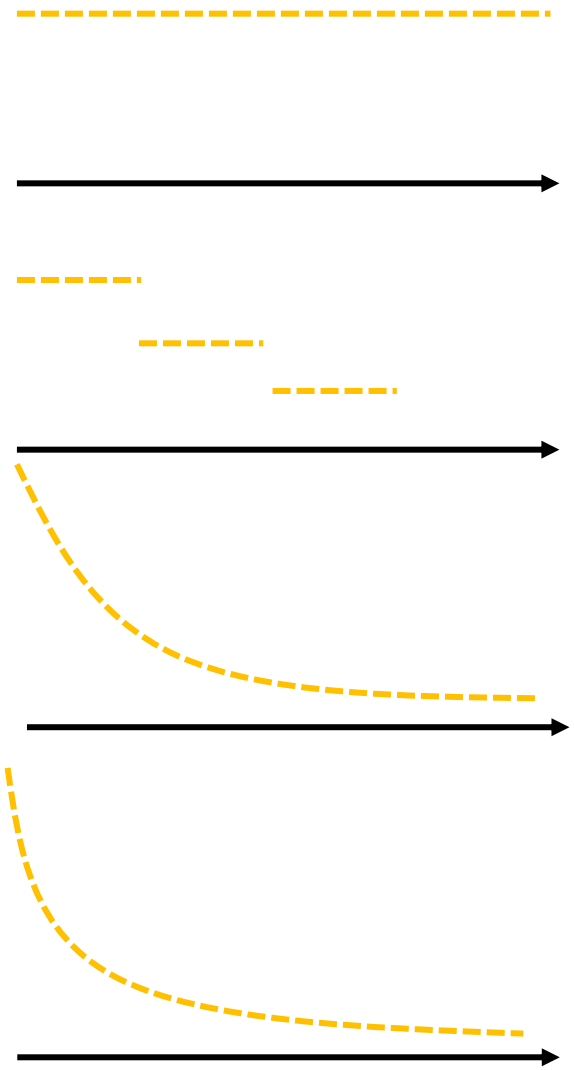
- The step sizes **theoretically** should satisfy the following [Robbins–Monro]

$$\sum_t^\infty \eta_t = \infty \quad \text{and} \quad \sum_t^\infty \eta_t^2 < \infty$$

- Intuitively,
  - The first term ensures that search will explore enough
  - The second term ensures convergence

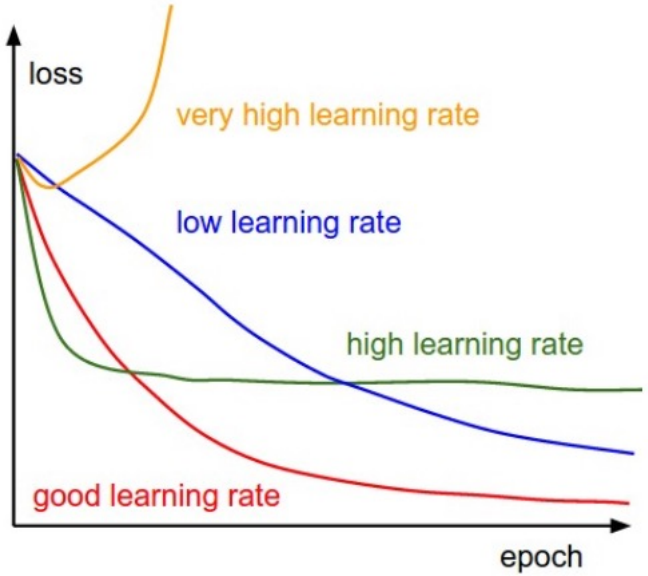
Learning rate schedules

- Constant
  - Learning rate remains the same for all epochs
- Step decay
  - Decrease every T number of epochs or when validation loss stopped decreasing
- Inverse decay $\eta_t = \frac{\eta_0}{1+\varepsilon t}$
- Exponential decay $\eta_t = \eta_0 e^{-\varepsilon t}$
- Cosine decay!
- Often step decay preferred
  - simple, intuitive, reusable, works well
- + warmup of learning rate common.



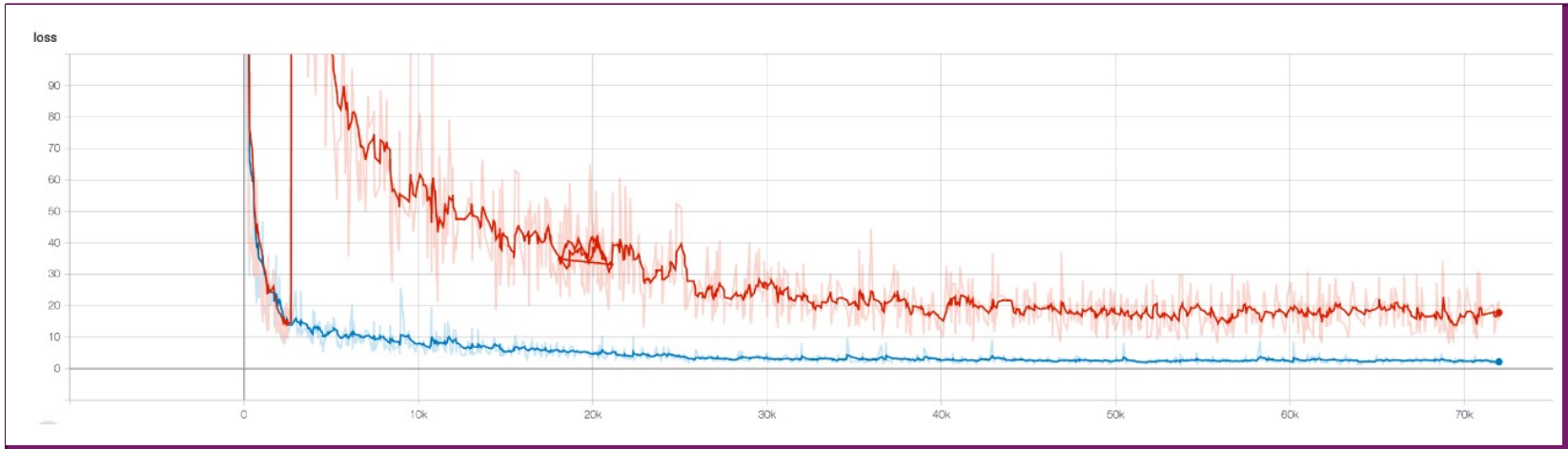
In practice

- Try several log-spaced values $10^{-1}, 10^{-2}, 10^{-3}, \ldots$ on a smaller set
  - Then, you can narrow it down from there around where you get the lowest **validation** error
- You can decrease the learning rate every T (e.g., 100) training set epochs
  - Although this highly depends on your data



Picture credit:
Stanford Course

Quiz



You run your model twice, once it gets the blue curve, once the red curve. What do you do?

1) Check the individual values of the gradients
2) Check batch at which loss exploded
3) Try increasing batch-size or lowering the learning rate
4) All of the above

https://stackoverflow.com/questions/58633177/why-theres-a-big-jump-up-of-the-loss-curve-during-the-training

Dropout rate

- Start with a relatively small rate, like 20-50%
  - If too high, your network will underfit
- With dropout you can also try larger neural networks

Batch size

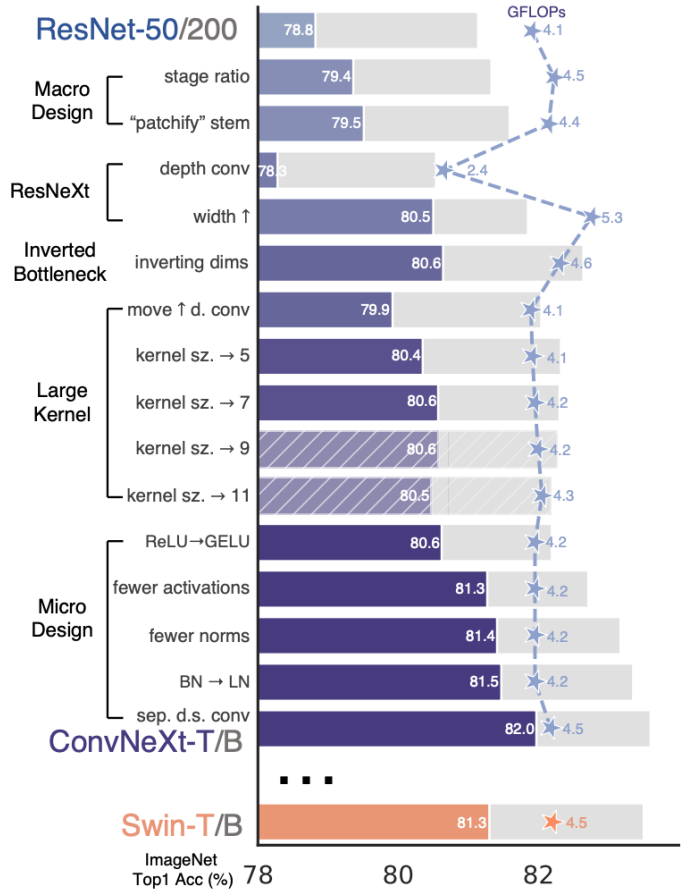
- If possible, start with at least 32
- Generally, as big as your GPU memory fits

Designing CNNs to become even better. (Don't try this at home)

Table 2: Ingredients and hyper-parameters used for ResNet-50 training in different papers. We compare existing training procedures with ours.

	Previous approaches					Ours		
Procedure → Reference	ResNet [13]	PyTorch [1]	FixRes [48]	DeiT [45]	FAMS (×4) [10]	A1	A2	A3
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	-	$0.02^{t/400}$	-	-	-
decay epochs	30	30	30	-	1	-	-	-
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	0.1	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

ResNet strikes back: An improved training procedure in timm. Wightman et al. 2021



A ConvNet for the 2020s. Liu et al. CVPR 2022

Number of layers and neurons

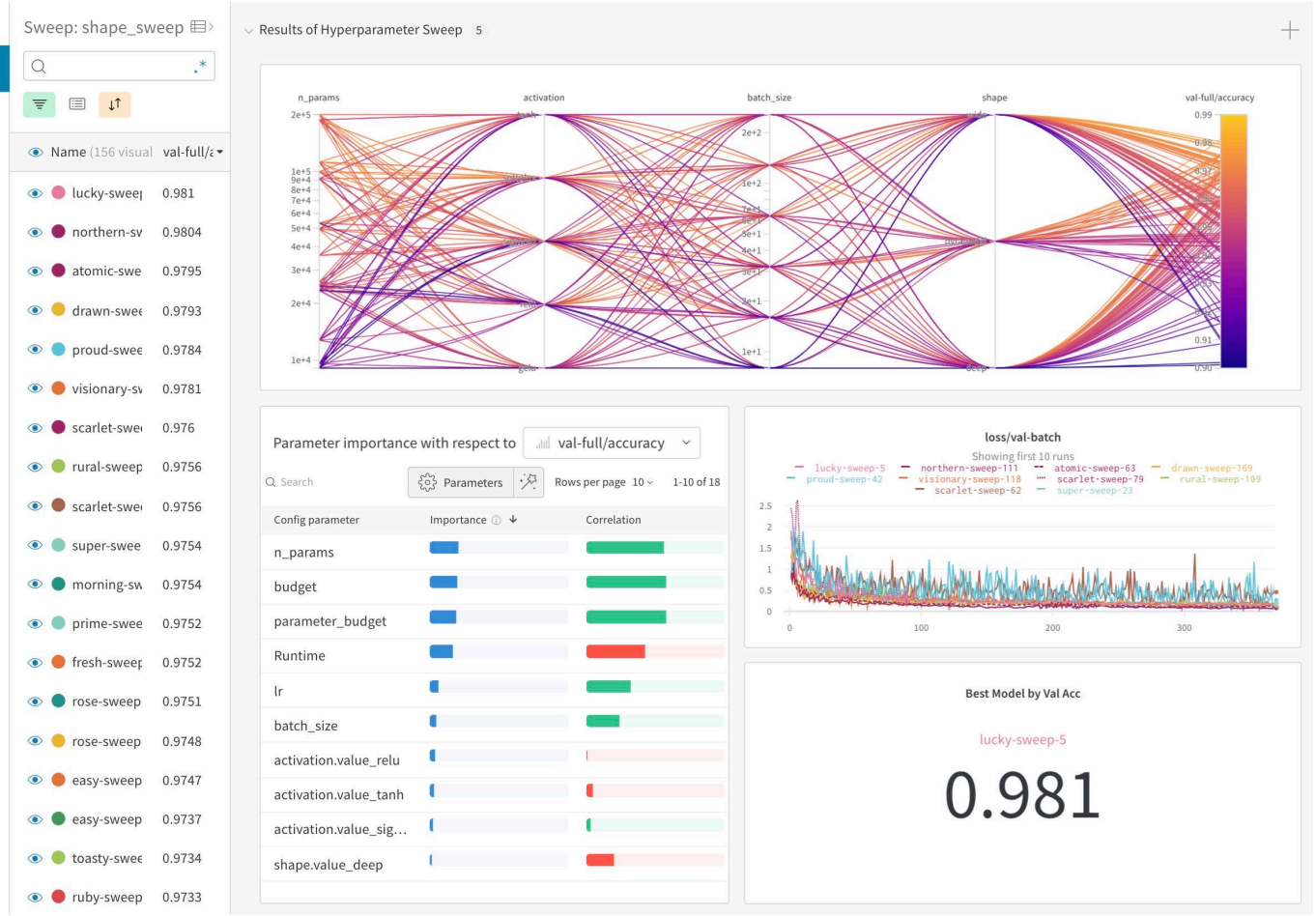
- For a new problem, generally start from moderate sizes
  - 3-5 layers
  - A few dozens neurons at most
  - When things check out, start increasing complexity
- For a known problem, *e.g.*, image classification, reuse hyperparameters
  - The one suggested by the model of choice are usually decent

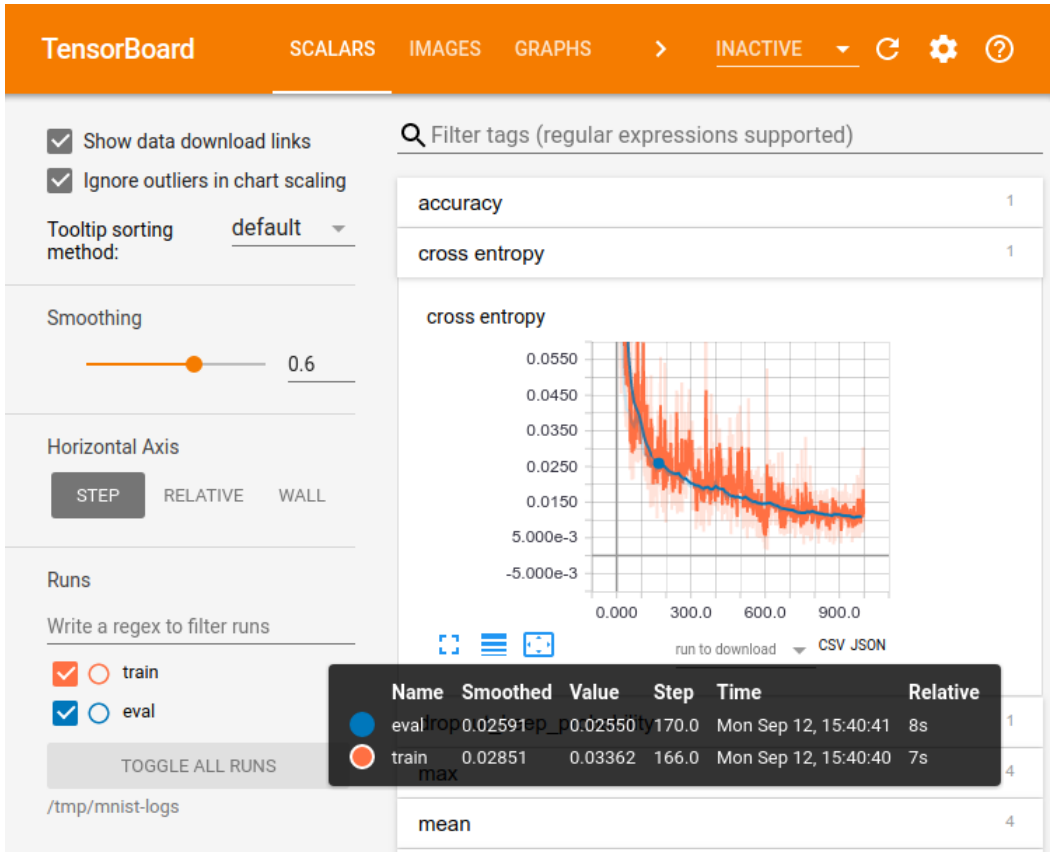
Generally: do not come up with new architectures. Use the existing ones and tailor to needs.

Babysitting Deep Nets

- Establish baselines
- Check that in the first round you get loss that corresponds to random guess
- Check network with few samples
  - Turn off regularization. You should predictably overfit and get a loss of 0
  - Turn on regularization. The loss should be higher than before
- **Always** a separate validation set for hyper-parameter tuning
  - Compare the training and validation losses - there should be a gap, not too large
- Preprocess the data (at least to have 0 mean)
- Initialize weights based on activations functions Xavier or Kaiming initialization
- Use regularization (ℓ_2-regularization, dropout, ...)
- Use batch normalization
- Prefer residual connections, they make a difference
- Use an experiment manager like tensorboard or Neptune or wand
- Track further metrics that are relevant to the problem at hand

Logging tools





Babysitting Deep Nets

- One of the most important skills
- Further reading:
  - http://karpathy.github.io/2019/04/25/recipe/
  - https://phillipi.github.io/6.s898/materials/slides/8_hackers_guide.pdf

Reading material

- Deep Learning Book: Chapter 8, 11
- Efficient Backprop
- How Does Batch Normalization Help Optimization?
- https://medium.com/paperspace/intro-to-optimization-in-deep-learning-momentum-rmsprop-and-adam-8335f15fdee2
- http://ruder.io/optimizing-gradient-descent/
- https://github.com/Jaewan-Yun/optimizer-visualization
- https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

- Advanced optimizers
- Initialization
- Normalization
- Regularization
- Hyperparameters

Reading material

- Chapter 8, 11
- And the papers mentioned in the slide