UTS Deep Learning – Regresi UTS Telkom (Tensorflow)

Nama : Hasan Al Banna

NIM : 1103223142

1. **Jika menggunakan model MLP dengan 3 hidden layer (256-128-64) menghasilkan underfitting pada dataset ini, modifikasi apa yang akan dilakukan pada arsitektur? Jelaskan alasan setiap perubahan dengan mempertimbangkan bias-variance tradeoff!**

   Hasil Analisa:

   Jika menggunakan MLP dengan 3 hidden layer (256-128-64) dan ternyata hasilnya underfitting (akurasi training rendah, model belum bisa nangkep pola dari data), berarti modelnya masih terlalu sederhana untuk dataset ini. Untuk mengatasinya, perlu menaikkan kapasitas model, tapi tetap harus mikirin soal bias-variance tradeoff, agar tidak menjadi overfitting.

   Modifikasi arsitektur dan alasannya:
   1. Tambah jumlah neuron per layer
   Misalnya dari:
   - 256 → 512
   - 128 → 256
   - 64 → 128

   Alasannya:
   Jika model underfit, itu tandanya model punya bias tinggi yang artinya terlalu sederhana untuk belajar hubungan kompleks dalam data. Menambah neuron membantu model menangkap pola yang lebih kompleks, dan menurunkan bias.

   2. Tambah jumlah hidden layer
   Misalnya jadi 4 atau 5 hidden layer (misal: 512-256-128-64-32)

   Alasannya:
   Tambah depth bisa bikin model belajar representasi hierarkis (fitur dari fitur), yang seringkali penting kalau data punya pola yang tidak cukup ditangkap layer dangkal. Ini juga bantu nurunin bias.

   3. Gunakan aktivasi non-linear yang lebih dalam
   Misal ganti ReLU biasa ke Leaky ReLU atau ELU

   Alasannya:
   Kadang aktivasi ReLU bisa bikin neuron “mati” (lihat dying ReLU), apalagi jika modelnya makin dalam. Aktivasi alternatif bisa membantu menjaga aliran gradien dan bikin training lebih stabil.

   Hati-hati sama variance!
   Jika menambah kapasitas model terlalu banyak, nanti bisa kebalikannya: model jadi overfitting (variance tinggi). Jadi untuk menjaga balance:

- Tambahkan dropout antar layer (misalnya 0.2–0.5).
- Gunakan batch normalization buat stabilisasi.
- Menggunakan early stopping agar training berhenti sebelum overfit.

2. **Selain MSE, loss function apa yang mungkin cocok untuk dataset ini? Bandingkan kelebihan dan kekurangannya, serta situasi spesifik di mana alternatif tersebut lebih unggul daripada MSE!**
   Jawaban:
   Selain MSE (Mean Squared Error), ada beberapa loss function lain yang bisa digunakan untuk dataset ini. Salah satunya MAE (Mean Absolute Error). MAE itu menghitung selisih absolut antara prediksi dan target, jadi gak terlalu "ngedrama" seperti MSE kalau ketemu outlier. Kelebihan MAE adalah dia lebih robust terhadap outlier, karena beda dikit sama beda jauh dianggap sama pentingnya. Tapi kekurangannya, MAE itu memberi gradien yang konstan, jadi kadang model lebih susah belajar, terutama saat awal-awal training.

   Alternatif lain yang lebih fleksibel itu Huber Loss. Dia seperti gabungan MSE dan MAE, di mana kalau error-nya kecil, dia seperti MSE (smooth dan bagus buat optimasi), tapi kalau error-nya besar, dia seperti MAE (gak terlalu dihukum keras). Jadi cocok banget digunakan oleh dataset ini yang memiliki banyak outlier disetiap fiturnya, tapi tetap pengen training yang stabil.
   Nah, loss-loss ini lebih unggul daripada MSE ketika kerja sama data yang noisy banget atau banyak outlier seperti dataset ini. Soalnya, MSE itu sensitif banget sama nilai ekstrem, outlier satu aja bisa narik loss jadi tinggi banget dan ngeganggu proses learning. Jadi, kalo datanya gak terlalu bersih atau distribusinya gak normal, MAE atau Huber Loss bisa jadi pilihan yang jauh lebih aman.

3. **Jika salah satu fitur memiliki range nilai 0-1, sedangkan fitur lain 100-1000, bagaimana ini memengaruhi pelatihan MLP? Jelaskan mekanisme matematis (e.g., gradien, weight update) yang terdampak!**
   Jawaban :
   Jika salah satu fitur nilainya cuma 0–1, tapi fitur lain punya range 100–1000, itu bisa membuat pelatihan MLP jadi tidak seimbang. Masalah utamanya ada di proses backpropagation dan update bobot (weight update) yang dipengaruhi banget sama nilai input dan gradien.

   Dampak secara matematis:
   1. Fitur yang besar akan dominasiin weight update
   Ingat rumus update bobot:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

   Nah, turunan loss terhadap bobot itu tergantung juga dari nilai input fitur. Jadi kalau satu fitur nilainya besar (misal 500), dan satu lagi kecil (0.5), fitur yang besar bakal hasilin gradien lebih besar, yang bikin bobotnya lebih banyak berubah tiap epoch.

Artinya: fitur kecil gak terlalu dipertimbangin pas training, walaupun mungkin sebenarnya penting!

2. Gradien bisa jadi gak stabil
Fitur dengan range yang jauh beda bisa bikin scale dari aktivasi antar neuron jadi berantakan. Apalagi di MLP yang punya banyak layer, ini bisa nyebabin:

- Vanishing gradient (gradien makin kecil di layer awal)
- Exploding gradient (gradien terlalu gede)

Keduanya bikin training lambat, gak stabil, atau malah gak belajar sama sekali.

3. Proses konvergensi jadi lebih lambat
Karena bobot terus disesuaikan buat mengejar fitur yang dominan, model butuh waktu lebih lama buat belajar representasi yang adil buat semua fitur. Kadang malah gak nemu solusi optimal karena stuck ngikutin fitur yang punya scale besar doang.

Solusinya apa?
Ya feature scaling. Biasanya menggunakan:

- Min-Max Scaling (biar semua fitur ada di range 0–1)
- Standardization (mean 0, std dev 1)

Tujuannya agar semua fitur punya skala yang setara, jadi gradiennya seimbang dan weight update lebih adil.

4. **Tanpa mengetahui nama fitur, bagaimana Anda mengukur kontribusi relatif setiap fitur terhadap prediksi model? Jelaskan metode teknikal (e.g., permutation importance, weight analysis) dan keterbatasannya!**
Jawaban :
Ketika gak tahu nama fiturnya, ada beberapa cara teknikal yang bisa dipakai buat mengukur kontribusi relatif fitur ke model. Ini penting banget buat mngerti model lo kerja gimana, apalagi kalau lo pengen transparansi atau explainability.

1. Permutation Importance
Cara kerja:

- Satu per satu fitur diacak (shuffle nilainya).
- Model dipakai buat prediksi lagi.
- Lihat seberapa turun performa model (misal: akurasi, MSE, AUC, dll).

Kalau performa anjlok pas satu fitur diacak, berarti fitur itu penting banget.
Kelebihan:

- Bisa dipakai di model apapun (MLP, CNN, XGBoost, dll).
- Gak peduli skala nilai fitur.

Kekurangan:

- Lambat (karena harus evaluasi model berulang kali).
- Kalau fitur saling berkorelasi, hasil bisa menyesatkan (fitur lain bisa "nambal" fitur yang diacak).

2. Weight Analysis (khusus model linear / shallow MLP)

Cara kerja:

- Fitur dengan bobot lebih besar (positif/negatif) diasumsikan lebih berpengaruh.

Kelebihan:

- Simpel, cepat, langsung dari model.

Kekurangan:

- Gak berlaku buat model nonlinear dalam (deep MLP, CNN).
- Bobot besar gak selalu berarti fitur penting kalau datanya kecil nilainya → interaksi bobot dan input matters.

3. SHAP (SHapley Additive exPlanations)

Cara kerja:

- Berdasarkan teori game, ngitung kontribusi setiap fitur terhadap prediksi, satu per satu.
- Anggap tiap fitur kayak "pemain" dalam tim yang nyumbang skor.

Kelebihan:

- Bisa kasih penjelasan per instance maupun rata-rata.
- Cocok buat model kompleks.

Kekurangan:

- Berat secara komputasi.
- Implementasinya butuh library tambahan (kayak SHAP di Python).

---

4. Integrated Gradients (khusus deep learning)

Cara kerja:

- Nge-track perubahan output model dari baseline input ke input sebenarnya.
- Gradien dikalikan dengan selisih input → hasilnya kontribusi fitur.

Kelebihan:

- Cocok buat model nonlinear.
- Bisa visualisasi kontribusi.

Kekurangan:

- Butuh akses ke gradien internal model.
- Lebih ribet implementasinya.

5. **Bagaimana Anda mendesain eksperimen untuk memilih learning rate dan batch size secara optimal? Sertakan analisis tradeoff antara komputasi dan stabilitas pelatihan!**

Jawaban :

Untuk menentukan learning rate dan batch size yang optimal, biasanya saya memulai dengan eksperimen learning rate finder. Teknik ini bantu nemuin range learning rate yang bagus dengan cara naikin learning rate secara bertahap selama beberapa iterasi awal, terus diplot antara loss dan learning rate-nya. Dari grafik itu kelihatan titik di mana loss mulai turun drastis sebelum akhirnya naik lagi, dan dari situ kita bisa ambil learning rate yang ideal. Setelah itu, baru gue coba-coba beberapa batch size misalnya 16, 32, 64, atau 128 sambil lihat dampaknya ke stabilitas loss dan performa validasi. Di sinilah muncul tradeoff-nya: batch size kecil biasanya bikin gradien lebih "noisy", tapi justru itu bisa bantu model keluar dari local minima dan generalisasinya jadi lebih bagus. Sebaliknya, batch size besar bikin training lebih stabil dan cepat (karena paralel di GPU), tapi kadang bisa bikin model overfit dan butuh

memori lebih besar. Learning rate juga punya tradeoff yang mirip: kalau terlalu besar, training bisa cepet tapi gak stabil dan gampang gagal konvergen, sedangkan kalau terlalu kecil, training bisa super lambat dan bisa-bisa stuck di local minima. Jadi, eksperimennya harus nyari titik tengah antara kecepatan komputasi dan stabilitas pelatihan, sambil terus monitor metrik validasi buat lihat apakah kombinasi yang dipilih benar-benar efektif.